Altera SoC Embedded Design Suite

User Guide

Subscribe
Send Feedback

ug-1137
2013.11.08

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Contents

Introduction to SoC Embedded Design Suite.....1-1
Overview.....1-1
Device Tree Binary.....1-2
Hardware/Software Development Roles.....1-3
Hardware – Software Development Flow.....1-5

Installing the Altera SoC Embedded Design Suite.....2-1
Installation Folders.....2-1
Installing the SoC EDS.....2-1
Installing the ARM DS-5 Altera Edition Toolkit.....2-2
Installing USB Serial Drivers.....2-2
Installing Altera USB Blaster Drivers.....2-2

Bare Metal Compiler.....3-1

Licensing.....4-1
Getting the License.....4-1
Activating the License.....4-2

Getting Started Guides.....5-1
Getting Started with Board Setup.....5-1
External Connections.....5-1
Dip Switch Settings.....5-2
Jumper Settings.....5-2
Getting Started with Running Linux.....5-2
Getting Started with Preloader.....5-3
Getting Started with Bare Metal Debugging.....5-6
Starting the Eclipse IDE.....5-6
Importing the Sample Application.....5-7
Compiling the Sample Application.....5-7
Running the Sample Application.....5-8
Getting Started with the Hardware Library.....5-11

HWLIB Sample Application Overview....5-11
Starting the Eclipse IDE....5-12
Importing the Sample Application....5-12
Compiling the Sample Application....5-14
Running the Sample Application....5-15
Getting Started with Peripheral Register Visibility....5-18
Getting Started with Linux Application Debugging....5-21
Configuring Linux....5-22
Starting Eclipse via the Embedded Command Shell....5-22
Importing the Sample Application....5-23
Compiling the Sample Application....5-24
Setting up Remote System Explorer....5-25
Running Sample Application....5-28
Getting Started with Linux Kernel and Driver Debugging....5-31
Prerequisites....5-32
Starting Eclipse via the Embedded Command Shell....5-32
Debugging the Kernel....5-32
Getting Started with Tracing....5-37
Getting Started with Cross Triggering....5-41
Enabling Cross-triggering on HPS....5-41
Enabling Cross-triggering on FPGA....5-43
Cross-triggering Prerequisites....5-44
FPGA Triggering HPS Example....5-46
HPS Triggering FPGA Example....5-49

ARM DS-5 Altera Edition....6-1
Starting Eclipse....6-1
Project Management....6-2
Debugging....6-4
Debug Configuration Options....6-7
DTSL Options....6-14

Embedded Command Shell....7-1

HPS Preloader User Guide....8-1
HPS Configuration....8-1
Preloader Support Package Generator....8-2

Hardware Handoff Files....8-3
Using the Preloader Support Package Generator GUI....8-3
Preloader Support Package Files and Folders....8-4
Command-Line Tools for the Preloader Support Package Generator....8-4
BSP Settings....8-8
Preloader Compilation....8-13
Configuring FPGA with Image from QSPI Flash....8-13
Loading Next Image from FAT Partition on SD Card....8-14
Preloader Image Tool....8-14
Operation of the Preloader Image Tool....8-14
Tool Usage....8-15
Output Image Layout....8-16
mkimage Tool....8-17
mkimage Tool Options....8-18
mkimage Tool Image Creation....8-19

Hardware Library Overview....9-1
Feature Description....9-2
SoC Abstraction Layer (SoCAL)....9-2
Hardware Manager (HW Manager)....9-2
Hardware Library Reference Documentation....9-3

HPS Flash Programmer User Guide....10-1
HPS Flash Programmer Command-Line Utilities....10-1
How the HPS Flash Programmer Works....10-2
Using the Flash Programmer from the Command Line....10-2
HPS Flash Programmer....10-2
HPS Flash Programmer Command Line Examples....10-4
Supported Memory Devices....10-5

Linux Compiler....11-1

Yocto Plugin....12-1

Device Tree Generator....13-1

Introduction to SoC Embedded Design Suite

1

Subscribe Send Feedback

The Altera SoC Embedded Design Suite provides the tools needed to develop embedded software for Altera's SoC devices.

The Altera® system on a chip (SoC) Embedded Design Suite (EDS) is a comprehensive tool suite for embedded software development on Altera SoC devices. The Altera SoC EDS contains development tools, utility programs, run-time software, and application examples that enable firmware and application software development on the Altera SoC hardware platform.

Overview

The Altera SoC EDS enables you to perform all required software development tasks targeting the Altera SoCs, including:

- Board bring-up
- Device driver development
- Operating system (OS) porting
- Bare-metal application development and debugging
- OS- and Linux- based application development and debugging
- Debug systems running symmetric multiprocessing (SMP)
- Debug software targeting soft IP residing on the FPGA portion of the device

The major components of the SoC EDS include:

- ARM® Development Studio 5 (DS-5™) Altera Edition Toolkit
- Compiler tool chains:
  - Bare-metal GNU Compiler Collection (GCC) tool chain from Mentor Graphics®
  - Linux GCC compiler tool chain from Linaro
- Pre-built Linux package including:
  - Linux kernel executable
  - Linux kernel U-boot image
  - Device tree blob
  - Secure Digital (SD) card image
  - Script to download Linux source code from the Git tree on the Rocketboards website (www.rocketboards.org). The script downloads the sources corresponding to the pre-built Linux package.
- SoC Hardware Library (HWLIB)

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

- Hardware-to-software interface utilities:
  - Preloader generator
  - Device tree generator
- Sample applications
- Golden Hardware Reference Design (GHRD) including:
  - FPGA hardware project
  - FPGA hardware SOF file
  - Precompiled preloader
- Embedded command shell, allowing easy invocation of the included tools
- Yocto Eclipse plugin
- Quartus II Programmer and SignalTap II

Note: The Linux package included in SoC EDS is not an official release and is intended to be used only as an example. Use the official Linux release described in the *Golden System Reference Design (GSRD) User Manual* available on the Rocketboards website (www.rocketboards.org) or a specific release from the Git trees located on the Gitweb page of the Rocketboards website (www.rocketboards.org) for development.

Note: The SoC EDS was tested only with the Linux release that comes with it. Newer Linux releases may not be fully compatible with this release of SoC EDS.

Note: The Golden Hardware Reference Design included with SoC EDS is not an official release, and is intended to be used only as an example. For development purposes, please use the official GHRD release described in the *GSRD User Manual* available on the Rocketboards website (www.rocketboards.org).

Related Information

Rocketboards website

For more information about the GHRD releases and to obtain the official Linux and GHRD releases, refer to the Rocketboards website.

Device Tree Binary

There are two device tree binary (DTB) files delivered as part of SoC EDS:

- *<SoC EDS Installation directory>***/embeddedsw/socfpga/prebuilt_images/socfpga_cyclone5.dtb**.

  This is a generic DTB file which does not have any dependency on soft IP. FPGA programming and bridge releasing are not required before Linux starts running using this DTB.

  This bare bones DTB is intended for customers interested in bringing up a new board or just wanting to simplify their boot flow until they get to the Linux prompt. If what is being developed or debugged does not involve the FPGA, it's better to remove the FPGA complexities.

- *<SoC EDS Installation directory>***/examples/hardware/cv_soc_devkit_ghrd/soc_system.dtb**.

  This DTB is based on Golden Hardware Reference Design (GHRD) design, which is part of the GSRD (Golden System Reference Design). As the GHRD does contain soft IPs, this DTB will notify Linux to load the soft IPs drivers. Therefore, the FPGA needs to be programmed and the bridges released before booting Linux.

Related Information

Rocketboards website

For more information about the GSRD, refer to the Rocketboards website.

Hardware/Software Development Roles

Depending on your role in hardware or software development, you need a different subset of the SoC EDS toolkit. The following table lists some typical engineering development roles, and indicates which tools each role typically requires.

Table 1-1: Hardware/Software Development Roles

Tool	Hardware Engineer	Bare Metal Developer	RTOS Developer	Linux Kernel and Driver Developer	Linux Application Developer
ARM DS-5 - Debugging	√	√	√	√	√
ARM DS-5 - Tracing		√	√	√	
ARM DS-5 - Cross Triggering		√	√	√	
Hardware Libraries		√	√	√	
Preloader Generator	√	√	√	√	
Flash Programmer		√	√	√	√
Bare Metal Compiler	√	√	√	√	
Linux Compiler				√	√
Yocto Plugin				√	√
Device Tree Generator				√	

This table lists typical tool usage, but your actual requirements depend on your specific project and organization.

Hardware Engineer

As a hardware engineer, you typically design the FPGA hardware in Qsys. You can use the ARM DS-5 Altera Edition's debugger to connect to the ARM cores and test the hardware. A convenient feature of the DS-5 debugger is the soft IP register visibility, using Cortex Microcontroller Software Interface Standard (CMSIS)

System View Description (**.svd**) files. With this feature, the hardware you can easily read and modify the soft IP registers from the ARM side.

As a hardware engineer, you might generate the Preloader for your hardware configuration. The Preloader is a piece of software that configures the HPS component according to the hardware design.

As a hardware engineer, you might also perform the board bring-up. You can use the ARM DS-5 debugger to verify that they can connect to the ARM and the board is working correctly.

These tasks require JTAG debugging, which is enabled only in the Subscription Edition. For more information , see the *Licensing* section.

Bare Metal and RTOS Developer

As either a bare metal or a RTOS developer, you need JTAG debugging and low-level visibility into the system.

Use the bare metal compiler to compile your code and the SoC Hardware Library to control the hardware in a convenient and consistent way.

Use the Flash Programmer to program the flash memory on the target board.

These tasks require JTAG debugging, which is enabled only in the Subscription Edition. For more information , see the *Licensing* section.

Linux Kernel and Driver Developer

As a Linux kernel or driver developer, you use the same tools as the RTOS developers, because you need low-level access and visibility into the system. However, you use the Linux compiler instead of the bare metal compiler. Youccan use the Yocto plugin to manage the project, and the device tree generator to generate device trees.

These tasks require JTAG debugging, which is enabled only in the Subscription Edition. For more information , see the *Licensing* section.

Linux Application Developer

As a Linux application developer, you write code that targets the Linux OS running on the board. Because the OS provides drivers for all the hardware, you do not need low-level visibility over JTAG. DS-5 offers a very detailed view of the OS, showing informaton such as which threads are running and which drivers are loaded.

You can use the Yocto plugin to manage the application build.

These tasks do not require JTAG debugging. You can perform them both in the Web and Subscription editions. For more information , see the *Licensing* section.

Related Information

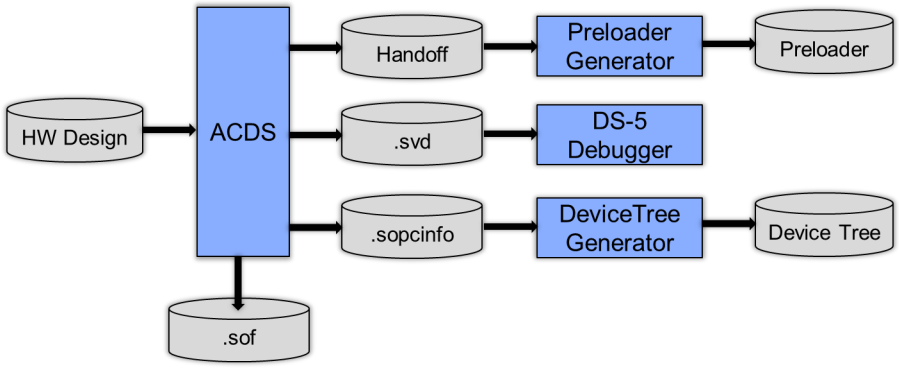
- **Licensing** on page 4-1
  For more information, refer to the Licensing section.
- **Licensing** on page 4-1
  For more information about **.svd** files, refer to the Hardware - Software Development Flow section.

Hardware – Software Development Flow

This section presents an overview of the design flow and the handoff between hardware and software development.

The Altera hardware-to-software handoff utilities allow hardware and software teams to work independently and follow their respective familiar design flows.

Figure 1-1: Altera Hardware-to-Software Handoff



The following handoff files are created when the hardware project is compiled:

- **Handoff** folder – contains information about how the HPS component is configured, including things like which peripherals are enabled, the pin muxing and IOCSR settings, memory parameters, etc
- **.svd** file – contains descriptions of the HPS registers and of the soft IP registers on FPGA side
- **.sopcinfo** file – contains a description of the whole system

The handoff folder is used by the preloader generator to create the Preloader.

The **.svd** file contains the description of the registers of the HPS peripheral registers and registers for soft IP components in the FPGA portion of the SoC. This file is used by the ARM DS-5 Debugger to allow these registers to be inspected and modified by the user.

SOPC Information (**.sopcinfo**) file, containing a description of the entire system, is used by the Device Tree Generator to create the Device Tree used by the Linux kernel.

Note: The soft IP register descriptions are not generated for all soft IP cores.

Related Information

- **HPS Preloader User Guide** on page 8-1
  For more information, refer to the HPS Preloader User Guide.
- **Device Tree Generator** on page 13-1
  For more information, refer to the Device Tree Generator section.

Installing the Altera SoC Embedded Design Suite

2

Subscribe Send Feedback

You must install the Altera® SoC Embedded Design Suite (EDS) and the ARM® Development Studio 5 (DS-5™) Altera Edition Toolkit to run the SoC EDS on an Altera SoC hardware platform.

Installation Folders

The default installation folder for SoC EDS is:

- c:\altera\13.1\embedded on Windows
- ~/altera/13.1/embedded on Linux

The default installation folder for Quartus Programmer is:

- c:\altera\13.1\qprogrammer on Windows
- ~/altera/13.1/qprogrammer on Linux

Throughout this document, the following are used:

- <Altera installation directory> to denote the location where Altera tools are installed
- <SoC EDS installation directory> to denote the location where SoC EDS is installed.

Installing the SoC EDS

Perform the following steps to install the SoC EDS Tool Suite in a Windows-based system:

1. Download the latest installation program from the **SoC Embedded Design Suite** page of the Altera website.
2. Run the installer to open the **Installing SoC Embedded Design Suite (EDS)** dialog box.
3. Click **Next** to start the Setup Wizard.
4. Accept the license agreement. Click **Next.**
5. Accept the default installation directory or browse to another installation directory. If you have previously installed the Quartus® II software, Altera recommends you accept the default SoC EDS installation directory to allow the Quartus II software and the SoC EDS Tool Suite to operate together. Click **Next**.
6. Select the components to be installed. Altera recommends that you install all components. Click **Next**.
7. The installer displays a summary of the installation. Click **Next** to start the installation process.
8. The installer displays the installation progress.
9. The installer displays a separate dialog box with the progress of the component installation.

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

10. When the installation is complete, turn on **Launch DS-5 Installation** to start the ARM DS-5 installation. Click **Finish**.

Note: On some Linux-based machines, you can install the SoC EDS with a setup GUI similar to the Windows-based setup GUI. Due to the variety of Linux distributions and package requirements, not all Linux machines can use the setup GUI. If the GUI is not available, you use an equivalent command-line process. Download the Linux installation program from the **SoC Embedded Design Suite** page on the Altera website.

Related Information

SoC Embedded Design Suite
For more information, refer to the *SoC Embedded Design Suite* page on the Altera website.

Installing the ARM DS-5 Altera Edition Toolkit

The ARM DS-5 Altera Edition Toolkit installer is started as the last step of the SoC EDS installation process.

1. A "Welcome" message is displayed. Click **Next**.
2. Accept the license agreement. Click **Next**.
3. Accept the default installation path, to ensure proper interoperability between SoC EDS and `ARM DS-5 AE`. Click **Next**.
4. Click **Install** to start the installation process. A progress bar is displayed.
5. A separate driver installation window is opened. Click **Next**.
6. Accept the driver installation and Click **Install**.
7. Drivers are installed successfully. Click **Finish**.
8. ARM DS-5 AE installation is completed. Click **Finish**.

Installing USB Serial Drivers

The Altera SoC development boards have dedicated devices that convert the RS232 to USB. The host machine requires dedicated drivers to use this feature.

Recent Linux distributions already have the necessary drivers included. The Windows version of the SoC EDS Installer comes with the necessary driver package in <*SoC EDS installation directory*>**\drivers\ftdi\CDM20830_Setup.exe**.

Run `CDM20830_Setup.exe`. Windows will install the drivers automatically when the USB cable is inserted.

Related Information

Installing the SoC EDS on page 2-1
For more information, refer to the Installing the SoC EDS section.

Installing Altera USB Blaster Drivers

The SoC EDS installation includes Quartus II Programmer and Signal Tap II applications. These applications connect to the board through an USB cable. Linux machines can access the USB connection directly, but dedicated drivers are required on a Windows host machine.

On Windows, the drivers are in the <*Altera installation directory*>**\qprogrammer\drivers** directory. Instruct Windows to install the drivers from that folder when the new hardware is detected.

Bare Metal Compiler 3

Subscribe Send Feedback

The bare metal compiler that is shipped with SoC EDS is the Mentor Sourcery CodeBench Lite Edition, version 4.6.3.

The compiler is a GCC based **arm-none- eabi** - port. That is, it targets the ARM processor, it assumes bare metal operation, and it uses the standard ARM EABI conventions.

The bare metal compiler is installed as part of the SoCEDS installation in the following folder: <*SoC EDS installation directory*>**host_tools/mentor/gnu/arm/baremetal**.

The Embedded Command Shell (opened by running the script from the SoC EDS installation folder) sets the correct environment PATH variables for the bare metal compilation tools to be invoked. That is, after starting the shell, commands like arm-none-eabi-gcc can be invoked directly. When the Eclipse environment is started from the Embedded Command Shell, it inherits the environment settings and it can call these compilation tools directly.

Alternatively, the full path to the compilation tools can be used: <*SoC EDS installation directory*>**/host_tools/mentor/gnu/arm/baremetal/bin.**

The bare metal compiler comes with full documentation, located in the following folder: <*SoC EDS installation directory*>**/host_tools/mentor/gnu/arm/baremetal/share/doc/arm-arm-none-eabi.** The documentation is offered in four different formats, to accommodate various user preferences:

- Html files
- Info files
- Man pages
- PDF files

Among the provided documents are:

- Compiler manual
- Assembler manual
- Linker manual
- Binutils manual
- GDB manual
- Getting Started Guide
- Libraries Manual

Related Information

Mentor Graphics

For more information on the Sourcery CodeBench Lite Edition and for downloading the latest version of the tools, if necessary; refer to the Mentor Graphics website (www.mentor.com).

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Licensing 4

Subscribe Send Feedback

The SoC EDS is available with three different licensing options:

- Subscription Edition
- Free Web Edition
- 30-day Evaluation of Subscription Edition

The only tool impacted by the selected licensing option is the ARM DS-5 Altera Edition. All the other tools offer the same level of features in all licensing options. This means the preloader generator, bare metal compiler, etc offer the same features no matter which licensing option is used.

The main difference between the licensing options consists in which types of debugging scenarios are enabled:

- Web Edition allows Linux Application Debugging over Ethernet
- Subscription Edition and 30-day Evaluation of the Subscription Edition allow:
  - JTAG-based Bare Metal Debugging
  - JTAG-based Linux Kernel and Driver Debugging
  - Linux Application Debugging over Ethernet

Related Information

SoC Embedded Design Suite

To download the SoC EDS tools and obtain more information about licensing, please refer to the SoC Embedded Design Suite page of the Altera website.

Getting the License

Depending on the licensing option, it is necessary to follow the steps detailed for each option to obtain the license.

Subscription Edition - If you have purchased the SoC EDS Subscription Edition, then you have already received an ARM license serial number. This is a 15-digit alphanumeric string with two dashes in between. You will need to use this number to activate your license in DS-5, as shown below.

Free Web Edition - For the free SoC EDS Web Edition, you will be able to use DS-5 perpetually to debug Linux applications over an Ethernet connection. Please get your ARM license activation code from the SoC Embedded Design Suite download page on the Altera website and then activate your license in DS-5, as shown below.

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

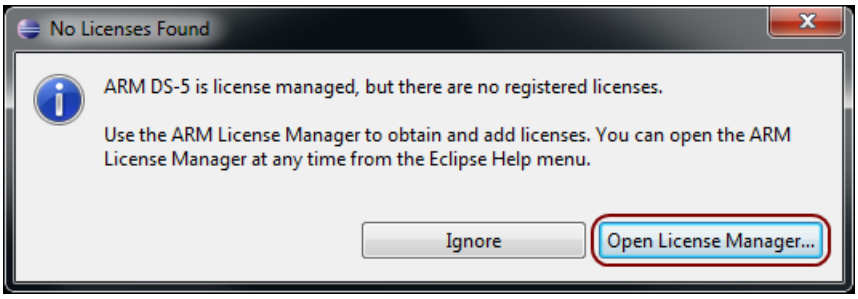
Activating the License

This section presents the steps required for activating the license in DS-5 Altera Edition by using the serial license number or activation code that were mentioned in the *"Getting the License"* section.

Note: An active user account is required to activate the DS-5 Altera Edition license. If you do not have an active user account, it can be created on the ARM *Self-Service* page available on the ARM website (**silver.arm.com**).

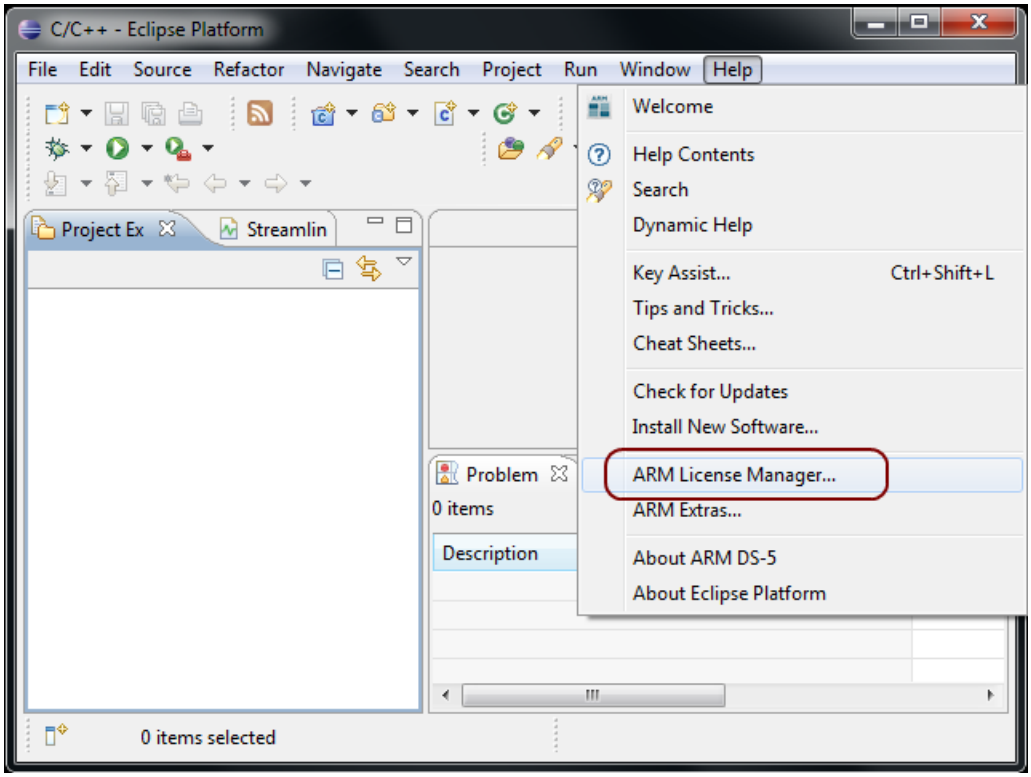
1. The first time the Eclipse IDE from the ARM DS-5 is run, it notifies you that it requires a license. Click the "Open License Manager" button.

Figure 4-1: No License Found


No Licenses Found
ARM DS-5 is license managed, but there are no registered licenses.
Use the ARM License Manager to obtain and add licenses. You can open the ARM License Manager at any time from the Eclipse Help menu.
Ignore
Open License Manager...

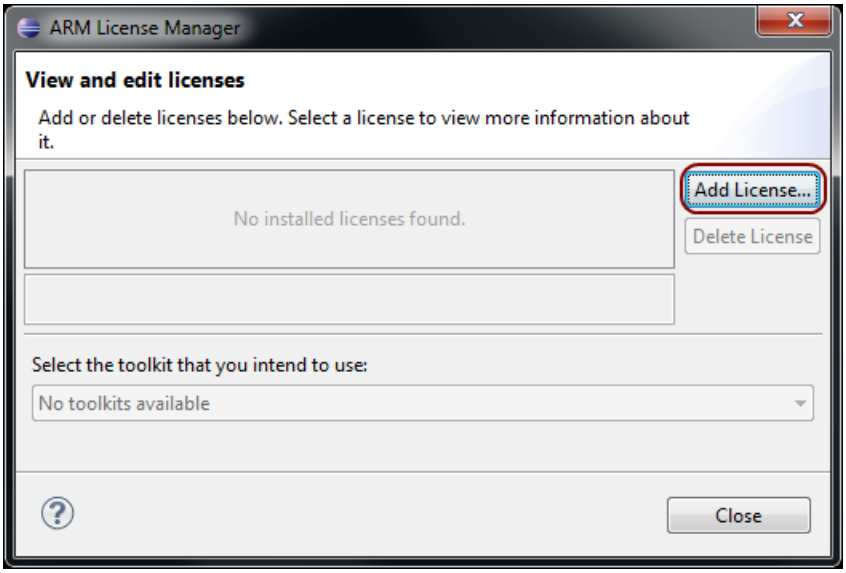
2. If at any time it is required to change the license, select **Help > ARM License Manager** to open the License Manager.

Figure 4-2: Accessing ARM License Manager



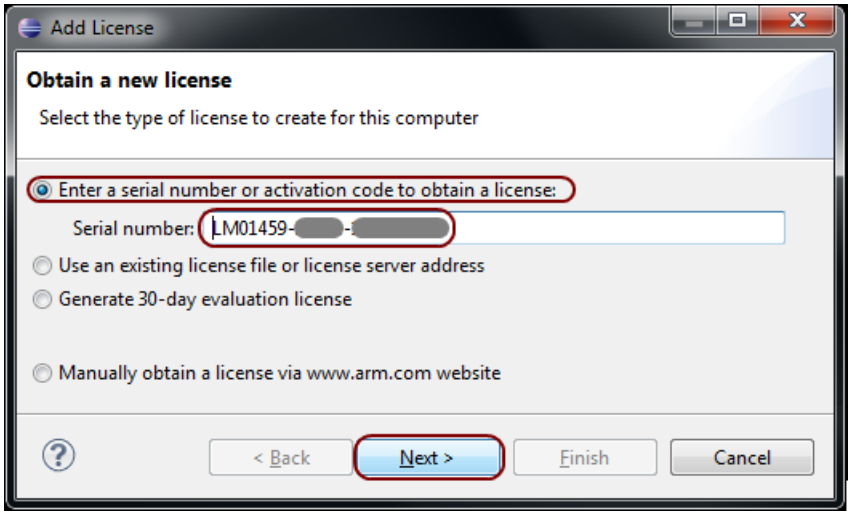
3. The **License Manager**, View and edit licenses dialog box opens and shows that a license is not available. Click the **Add License** button.

Figure 4-3: ARM License Manager


ARM License Manager
View and edit licenses
Add or delete licenses below. Select a license to view more information about it.
No installed licenses found.
Add License...
Delete License
Select the toolkit that you intend to use:
No toolkits available
Close

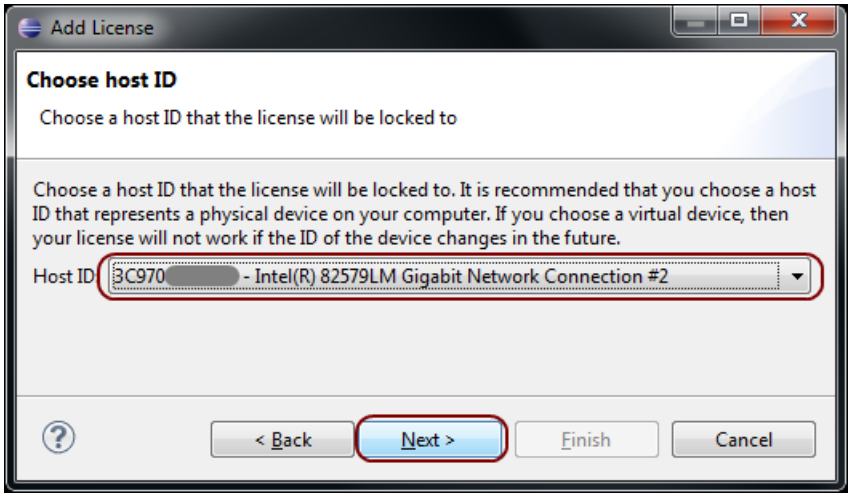
4. In the **Add License**, Obtain a new licenses dialog box, select the type of license to enter. In this example, select the radio button, **"Enter a serial number or activation code to obtain a license"** to enter the choices listed, below. When done, click **Enter**.
   a. ARM License Number for **Subscription Edition**.
   b. ARM License Activation Code for **Web Edition** and **30-Day Evaluation**.

Figure 4-4: Add License


Add License
Obtain a new license
Select the type of license to create for this computer
Enter a serial number or activation code to obtain a license:
Serial number:
LM01459-
Use an existing license file or license server address
Generate 30-day evaluation license
Manually obtain a license via www.arm.com website
< Back
Next >
Finish
Cancel

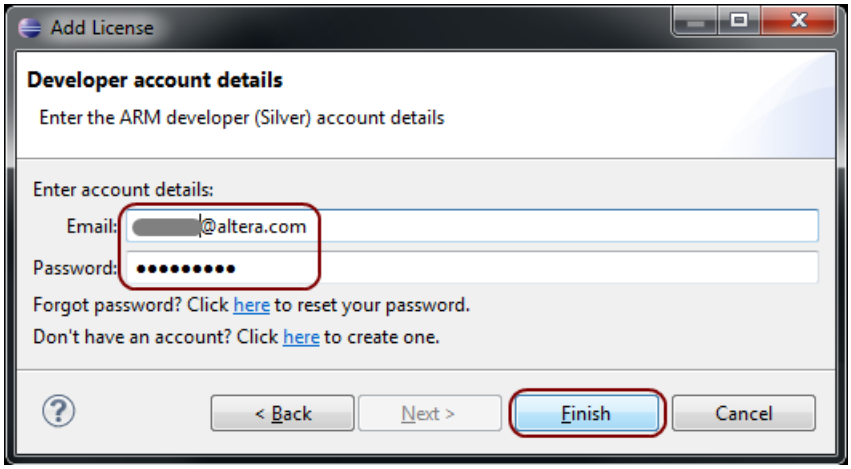
5. Click **Next**.
6. In the **Add License**, Choose Host ID dialog box, select the Host ID (Network Adapter MAC address) to tie the license to. If there are more than one option, select the one you desire to lock the license to. Click **Next**.

Figure 4-5: Add License


Add License
Choose host ID
Choose a host ID that the license will be locked to
Choose a host ID that the license will be locked to. It is recommended that you choose a host ID that represents a physical device on your computer. If you choose a virtual device, then your license will not work if the ID of the device changes in the future.
Host ID
3C970
- Intel(R) 82579LM Gigabit Network Connection #2
< Back
Next >
Finish
Cancel

7. In the **Add License**, Developer account details dialog box, enter an ARM developer (Silver) account. If you do not have an account, it can be created easily by clicking the provided link. After entering the account information, Click **Finish**.

Figure 4-6: Add License



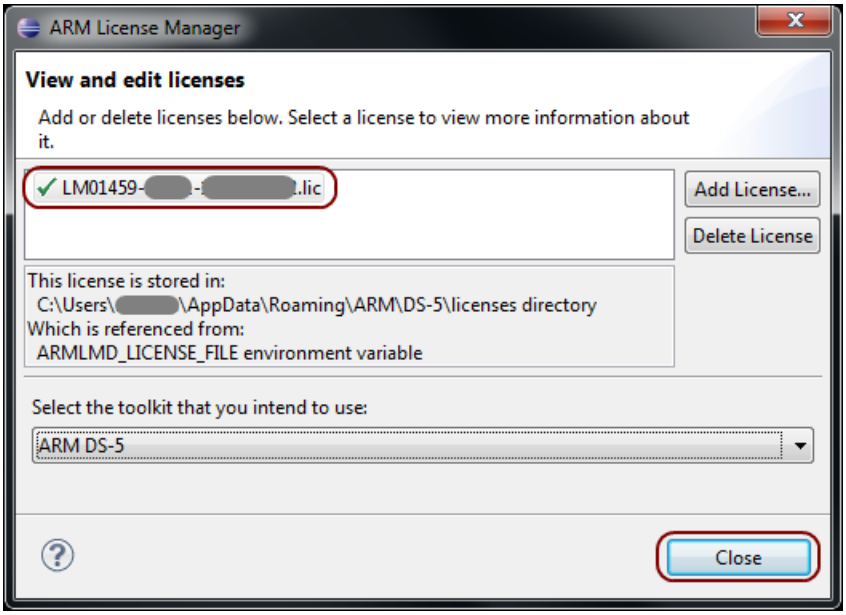
Note: The License Manager needs to be able to connect to the Internet in order to activate the license. If you do not have an Internet connection, you will need to write down your Ethernet MAC address, and generate the license directly from the *ARM Self-Service* web page on the ARM website (**silver.arm.com**), then select the "Already have a license" option in the License Manger.

Note: Only the Subscription Edition, with an associated license number can be activated this way. The Web Edition and Evaluation edition are based on activation codes, and these codes cannot be used on the *ARM Self-Service* web page on the ARM website (**silver.arm.com**). They need to be entered directly in the License Manager; which means an Internet connection is a requirement for licensing.

The ARM License Manager uses the Eclipse settings to connect to the Internet. The default Eclipse settings is to use the system-wide configuration for accessing the Internet. In case the License Manager cannot connect to the Internet, you can try to change the Proxy settings by going to **Window > Preferences > General > Network Connections**.

8. After a few moments, the `ARM DS-5` will activate the license and display it in the **License Manager**. Click **Close**.

Figure 4-7: ARM License Manager



Related Information

ARM Self-Service

For more information about creating user accounts, refer to the ARM website (silver.arm.com).

Getting Started Guides 5

Subscribe Send Feedback

This chapter presents a series of getting started guides aimed at enabling you to quickly get accustomed to doing the basic SoC software development tasks.

The following items are covered:

- Preloader
- Bare-Metal debugging
- SoC Hardware library (HWLIB)
- Peripheral register visibility
- Linux application debugging
- Linux Kernel and driver debugging
- Tracing
- Cross Triggering

The following additional topics are covered to support the above scenarios:

- Board setup – needed for all the scenarios
- Running Linux – needed for the scenarios that use Linux

The guides presented in this chapter are intedned to be run on a Cyclone V SoC Development board.

Getting Started with Board Setup

This section presents the necessary Altera Cyclone V Development Kit board settings in order to run Linux and the Getting Started examples.

External Connections

- External 19V power supply connected to J22 – DC Input
- Mini USB cable connected from host PC to J37 – Altera USB Blaster II connector. This is used for connecting the host PC to the board for debugging purposes.
- Mini USB cable connected from host PC to J8 – UART USB connector. This is used for exporting the UART interface to the host PC.
- Ethernet cable from connector J3 to local network. This is used if Linux network connectivity is desired.

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Dip Switch Settings

- SW1 = all switches OFF
- SW2 = all switches OFF
- SW3 = ON-OFF-OFF-OFF-ON-ON. This selects the proper FPGA configuration option (MSEL).
- SW4 = OFF-OFF-ON-ON. This selects both HPS and FPGA to be in the JTAG scan chain.

Jumper Settings

Number	Name	Setting
J5	9V	Open
J6	JTAG_HPS_SEL	Shorted
J8	JTAG_SEL	Shorted
J9	UART Signals	Open
J13	OSC1_CLK_SEL	Shorted
J15	JTAG_MIC_SEL	Open
J26	CLKSEL0	2-3 Shorted
J27	CLKSEL1	2-3 Shorted
J28	BOOTSEL0	2-3 Shorted
J29	BOOTSEL1	2-3 Shorted
J30	BOOTSEL2	1-2 Shorted
J31	SPI_I2C	Open

Getting Started with Running Linux

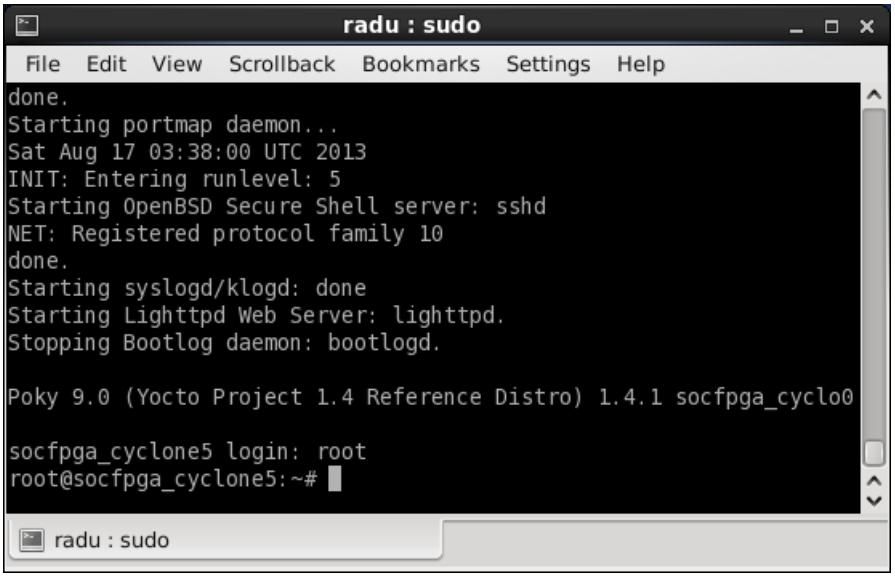
This section presents how to run the provided Linux image on the board, to be able to run the Getting Started sections related to Linux.

Note: The provided Linux image is an example only; use the latest version from the Rocketboards website (www.rocketboards.org) for your development.

The steps are:

1. Setup the board as described in Board Setup section.
2. Extract the SD card image from the archive <*SoC EDS installation directory*>**\embeddedsw\socfpga\prebuilt_images\sd_card_linux_boot_image.tar.gz**. The file is named **sd_card_linux_boot_image.img**. The command `tar -xzf<filename>` can be used from Embedded Command Shell to achieve this.
3. Write the SD card image to a micro SD card using the free tool **Win32DiskImager** from the Sourceforge Projects website (sourceforge.net) on Windows or the **dd** utility on Linux.
4. Power up the board using the **PWR** switch.
5. Connect a serial terminal from the host PC to the serial port corresponding to the UART USB connection; and use 115,200 baud, no parity, 1 stop bit, no flow control settings.
6. After successful boot, Linux will ask for the login name. Enter **root** and click **Enter**.

Figure 5-1: Linux Booted


radu : sudo
File Edit View Scrollback Bookmarks Settings Help
done.
Starting portmap daemon...
Sat Aug 17 03:38:00 UTC 2013
INIT: Entering runlevel: 5
Starting OpenBSD Secure Shell server: sshd
NET: Registered protocol family 10
done.
Starting syslogd/klogd: done
Starting Lighttpd Web Server: lighttpd.
Stopping Bootlog daemon: bootlogd.
Poky 9.0 (Yocto Project 1.4 Reference Distro) 1.4.1 socfpga_cyclo0
socfpga_cyclone5 login: root
root@socfpga_cyclone5:~#
radu : sudo

Related Information

- **Rocket Boards**
  For more information about the latest Linux version, refer to the Rocketboards website.
- **Sourceforge Projects**
  To obtain the free tool - **Win32DiskImager**, refer to the Projects section of the Sourceforge website.

Getting Started with Preloader

This section presents an example of how to generate and compile the Preloader for the Cyclone V SoC Golden Hardware Reference Design (GHRD) that is provided with SoC EDS.

The Preloader is an essential tool for SoC software. It performs the low-level initialization, brings up SDRAM memory, loads the next boot stage from flash to SDRAM and executes it.

The Preloader is already delivered as part of the GHRD in the <*SoC EDS installation directory*>**/examples/hardware/cv_soc_devkit_ghrd/software/preloader** folder.

In this example, you will re-create the **Preloader** in the folder <*SoC EDS installation directory*>**/examples/hardware/cv_soc_devkit_ghrd/software/spl_bsp**.

The screen snapshots presented in this section were created using the Windows version of SoC EDS, but the example can be run in a very similar way on a Linux host PC.

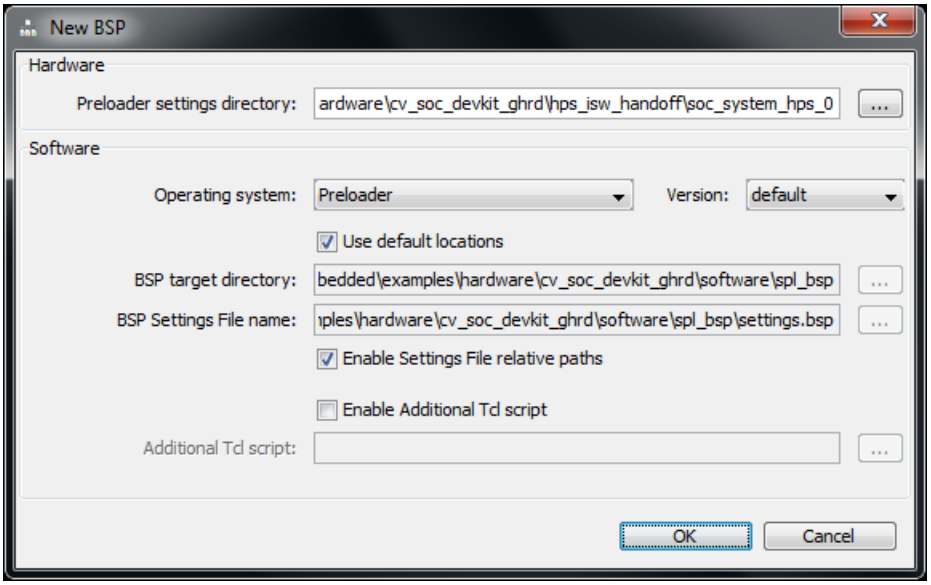
The steps to create the Preloader are:

1. Start an Embedded Command Shell by executing <*SoC EDS installation directory*>**\Embedded_Command_Shell.bat.**
2. Run the command, `bsp-editor`. The **BSP Editor** dialog box appears.

   Note: The tool that generates a preloader support package is the BSP Editor, also used to generate BSPs for other Altera products.

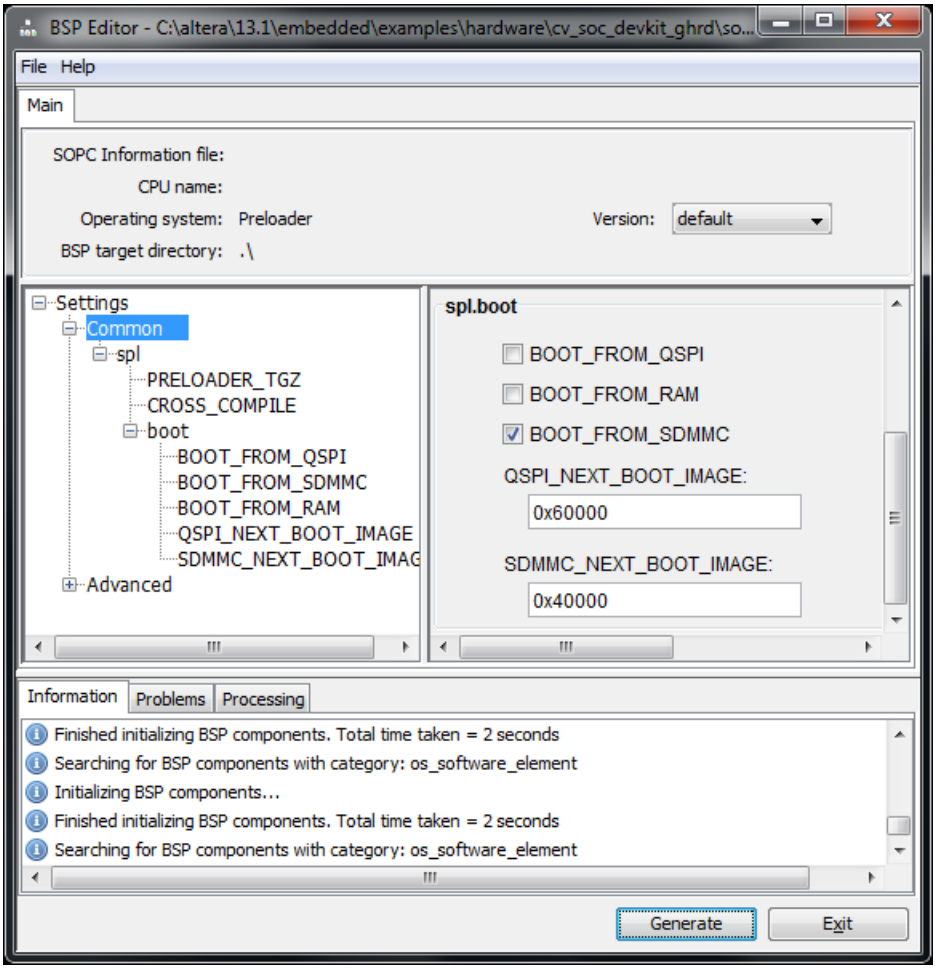
3. Select **File > New BSP**.The **New BSP** dialog opens.
4. Click the "..." button to browse for the Preloader settings directory in the **New BSP** dialog box.
5. Browse **<SoCEDS folder>\examples\hardware\cv_soc_devkit_ghrd\hps_isw_handoff\soc_system_hps_0** for the hardware handoff folder. The rest of the Preloader settings are populated automatically.

Figure 5-2: Populated Options in the New BSP window



6. Click **OK** to close the **New BSP** dialog box. This will populate the **BSP Editor** dialog box with the default settings.

Figure 5-3: Default Options in the BSP Editor window



7. Click **Generate** in the **BSP Editor** dialog box to generate the Preloader files.
8. Click **Exit** in the **BSP Editor** dialog box to exit the application.
9. In the Embedded Command Shell, execute the following commands:
   - cd <*SoC EDS installation directory*>**\examples\hardware\cv_soc_devkit_ghrd\software\spl_bsp**
   - **make**
10. The Preloader is ready to be used in the above folder. Some of the more relevant files that are created:
    - **preloader-mkpimage.bin** – Preloader with the proper header to be loaded by BootROM
    - **uboot-socfpga \ spl \ u-boot- spl** – Preloader ELF file, to be used for debugging purposes
    - **uboot-socfpga \tools\mkimage.exe** – Utility to add the header needed by the Preloader to recognize the next boot stage

Related Information

- **Preloader**
  For more information about the Preloader, refer to the *Preloader* section.

- **Booting and Configuration**
  For more information about Booting and Configuration with regards to Preloader, refer to the *Booting and Configuration* appendix in volume 3 of the *Cyclone V Device Handbook*.
- **Booting and Configuration**
  For more information about Booting and Configuration with regards to Preloader, refer to the *Booting and Configuration* appendix in volume 3 of the *Arria V Device Handbook*.

Getting Started with Bare Metal Debugging

The ARM DS-5 Altera Edition provides very powerful bare metal debugging capabilities.

This section presents running the ARM DS-5 Altera Edition for the first time, importing, compiling and running the Hello World bare-metal example application provided as part of SoC EDS.

Sample Application Overview

The provided sample application prints a "Hello" message on the debugger console, by using semihosting. This way no pins are used and all communication happens through JTAG.

The application is located in the 64 KB On-Chip RAM, and therefore does not require the SDRAM memory on the board to be configured.

Due to its simplicity and the fact that does not require pins or external resources to be configured, this application can be run on any board supporting the SoC device.

Note: Make sure that Linux (or another OS) is not running on the board prior to doing this example. An OS can interfere with the feature of downloading and debugging bare-metal applications.

Note: The screen snapshots and commands presented in this section were created using the Windows version of SoCEDS, but the example can be run in a very similar way on a Linux host PC.

Related Information

- **ARM DS-5 Altera Edition** on page 6-1
  For more information, refer to the *ARM DS-5 Altera Edition* section.
- **ARM DS-5 Documentation**
  The ARM DS-5 Altera Edition reference material can be accessed online on the documentation page of the ARM website (www.arm.com); and from Eclipse by navigating to **Help > Help Contents > ARM DS-5 Documentation**.

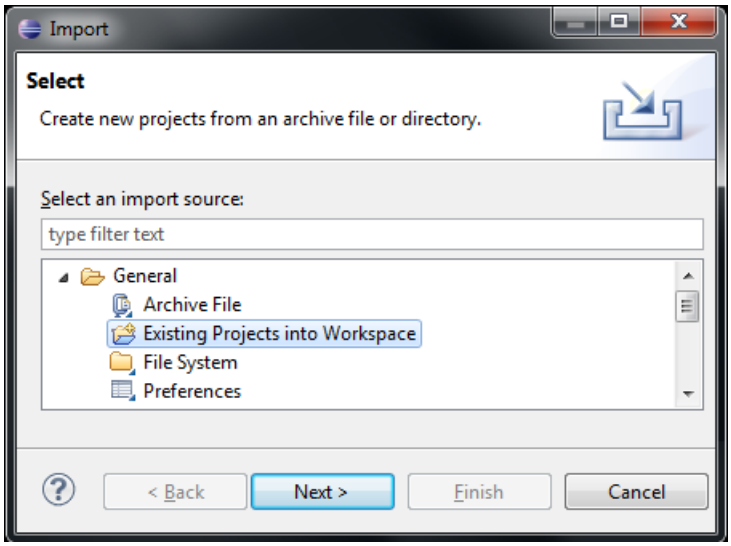
Starting the Eclipse IDE

1. Select **Start Menu > Programs > ARM DS-5 > Eclipse for DS-5** to start Eclipse. Alternatively, you can run `eclipse` command from the **Embedded Command Shell**.
2. The Eclipse tool, part of ARM DS-5 AE, prompts for the workspace folder to be used. Use the suggested folder and click **OK**.
3. The ARM DS-5 AE "Welcome" screen appears. It is instructive, and can be used to access documentation, tutorials and videos.

Importing the Sample Application

1. In Eclipse, select **File > Import**. The **Import** dialog box displays.
2. In the **Import** dialog box, select **General > Existing Projects into Workspace** and click **Next.** This will open the **Import Projects** dialog box.

Figure 5-4: Import Existing Project



3. In the **Import Projects** dialog box, select the **Select Archive File** option.
4. Click **Browse**, then navigate to *<SoC EDS installation directory>***\embedded\examples\software**, select the file **Altera-SoCFPGA-HelloWorld-Linux-GNU.tar.gz** and click **OK**.
5. Click **Finish**. The project is imported. The project files are displayed in the **Project Explorer** panel. The following files are part of the project:

Table 5-1: Project Files

File Name	Description
hwlib.c	Sample application source code
Altera-SoCFPGA-HelloWorld-Baremetal-GNU-Debug.launch	Launcher file used to run/debug the sample application from within Eclipse
altera-socfpga-hosted.ld	Linker script
debug-hosted.ds	Debugger script use to load the sample application
Makefile	Makefile used to compile the sample application

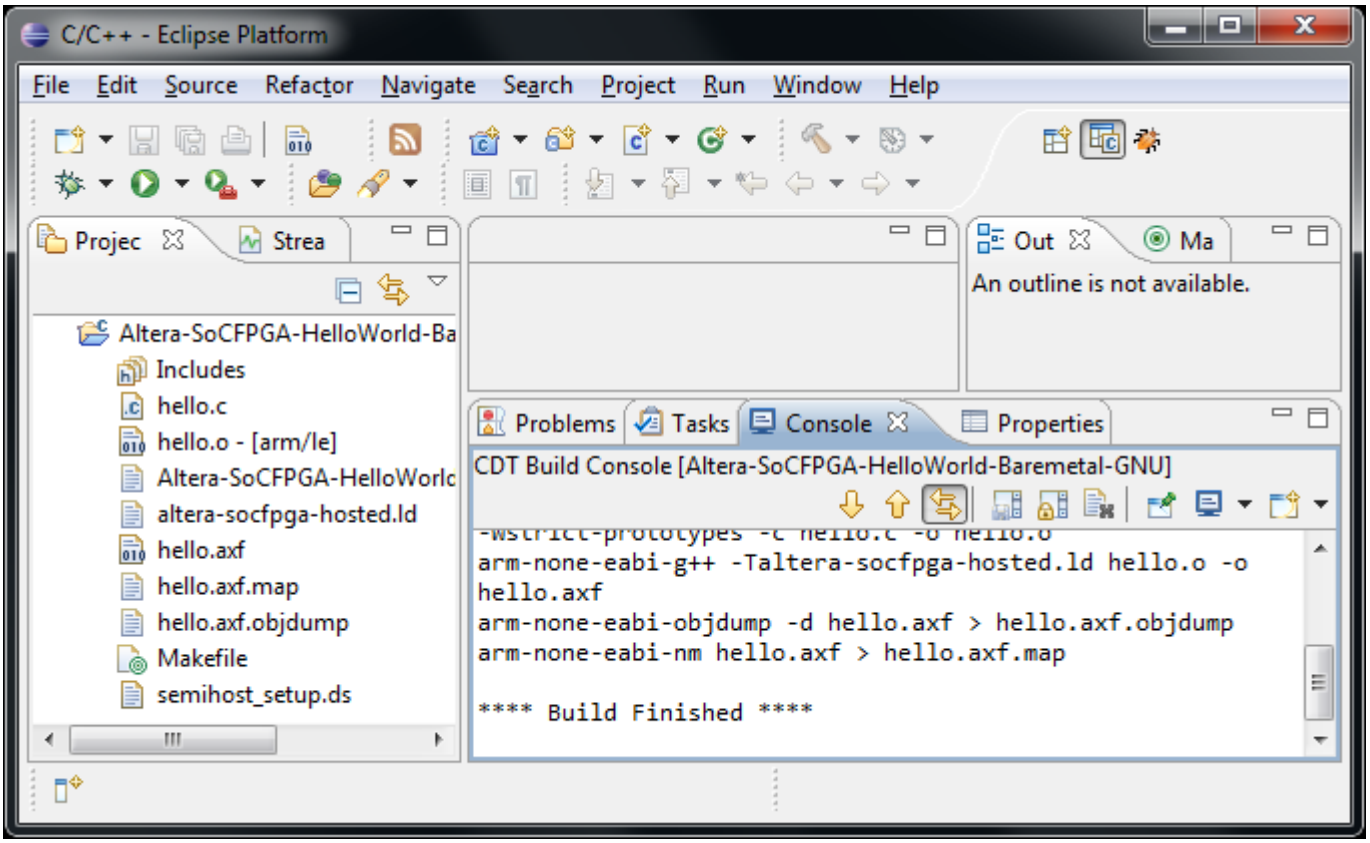
Compiling the Sample Application

The sample application is compiled using the Mentor bare-metal GCC tool chain invoked by the Makefile.

1. To compile the application, select the project in **Project Explorer**.

2. Select **Project** > **Build Project**.

Figure 5-5: Project Compiled



3. The project compiles and the **Project Explorer** shows the newly created **hwlib.axf**executable file as shown in the above figure. The **Console** dialog box shows the commands that were executed.

Running the Sample Application

Before running the sample application, perform the following setup:

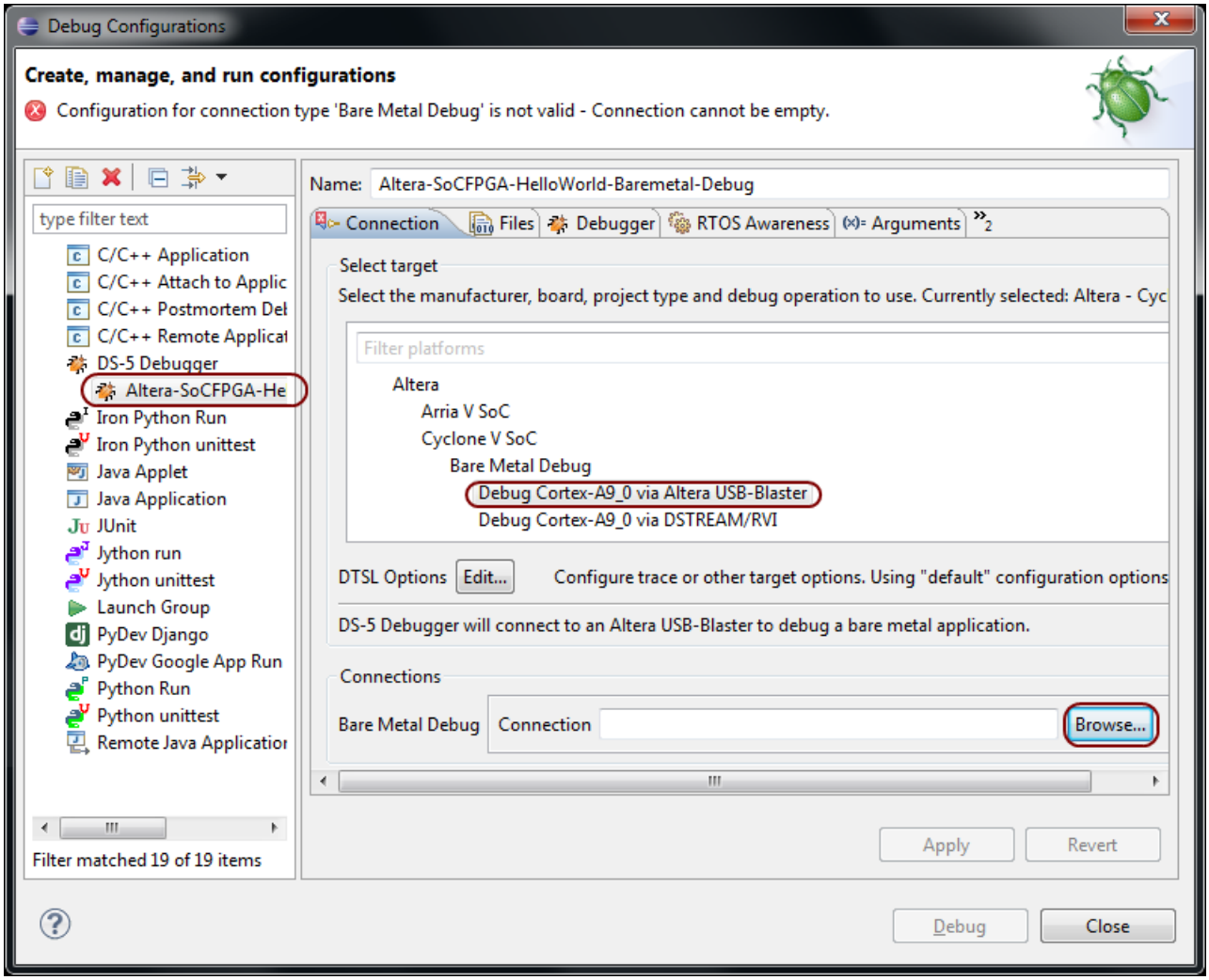
- Setup the board as described in **Getting Started with Board Setup**
- Connect mini USB cable from DevKit board connector J37 to PC
- Connect 19V power supply to the DevKit
- Turn on the board using the **PWR** switch

1. Select **Run** > **Debug Configurations..** to access the launch configurations. The sample project comes with a pre-configured launcher that allows the application to be run on the board.
2. In the **Debug Configurations** dialog box, on the left panel, select **DS-5 Debugger > Altera-SoCFPGA-HelloWorld-Baremetal-Debug**.

   The **Target** is already pre-configured to be **Altera > Cyclone VSoC > Bare Metal Debug > Debug Cortex-A9_0 via Altera USB-Blaster**.

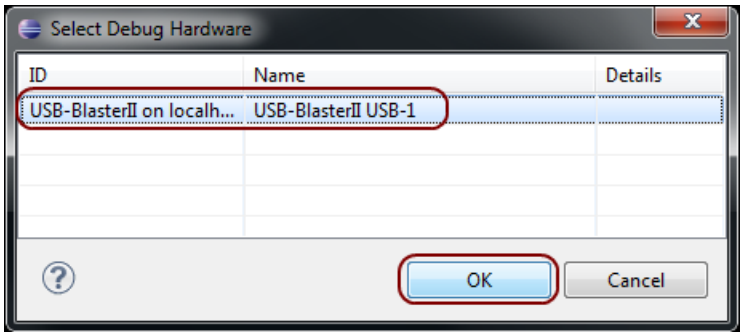
3. Click **Browse** to select the USB Blaster connection.

Figure 5-6: Debug Configuration



4. In the **Select Debug Hardware** dialog box, select the desired USB Blaster and click **OK**.

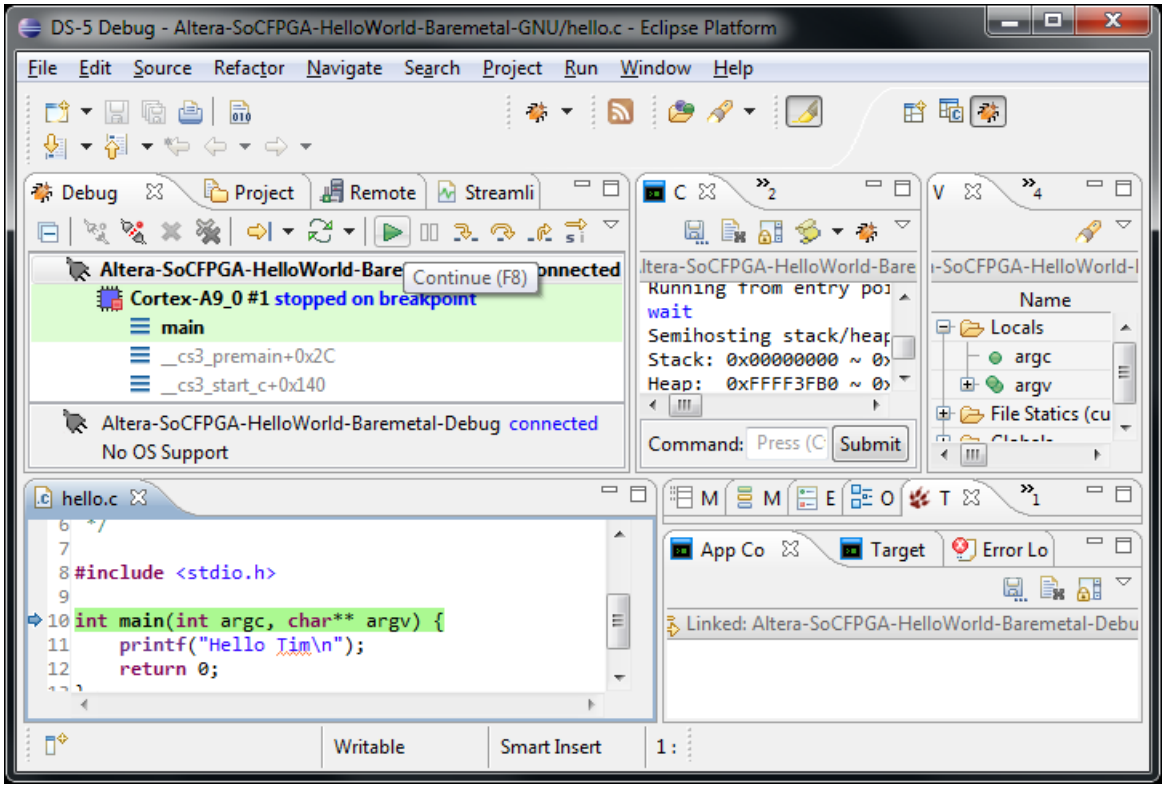
Figure 5-7: Select Debug Hardware



5. Click the **Debug** button from the bottom of the **Debug Configurations** dialog box.
6. Eclipse ask whether to switch to **Debug Perspective**. Click **Yes** to accept it.
   The debugger downloads the application on the board through JTAG, enables semi-hosting using the provided script, and runs the application until the PC reaches the **main** function.

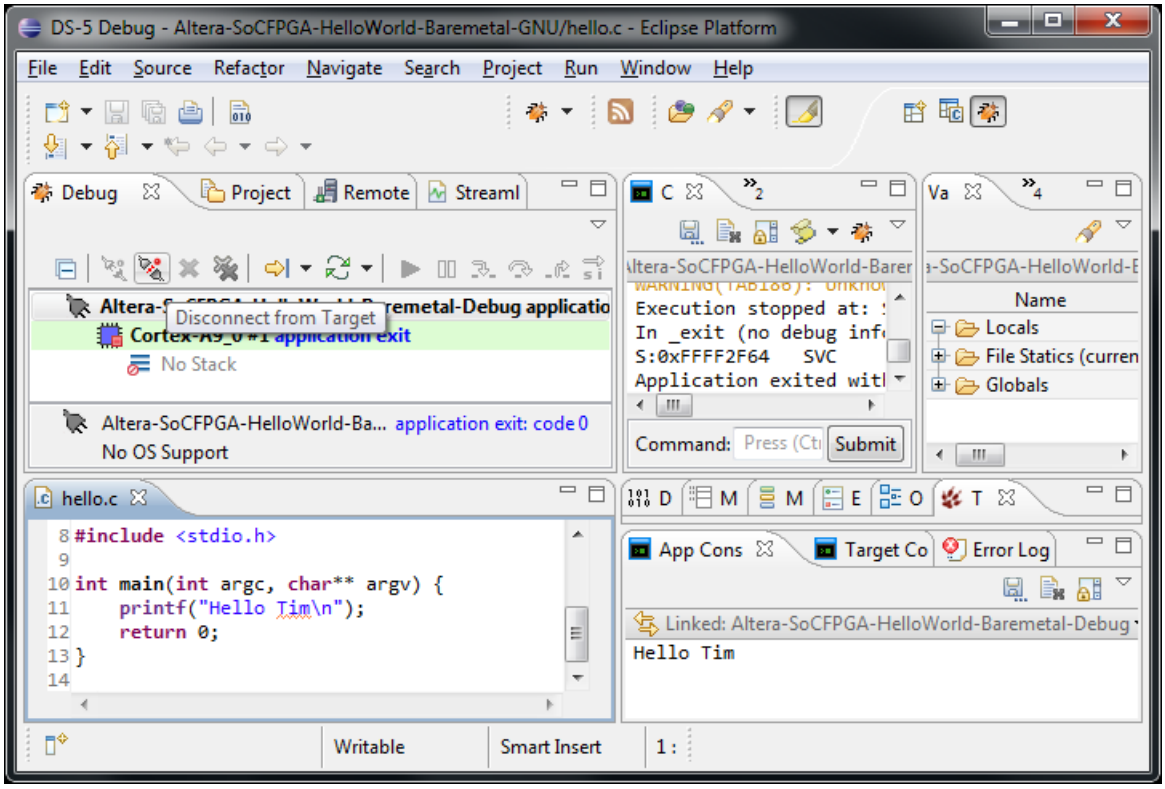
   At this stage, all the debugging features of DS-5 can be used: viewing and editing registers and variables, looking at the disassembly code.

Figure 5-8: Program Downloaded



7. Click **Continue** green button (or press **F8**) to run the application. It displays the hello message in the **Application Console**.

Figure 5-9: Debugging Session window



8. Click **Disconnect from Target** button to close the debugging session.

Getting Started with the Hardware Library

The SoC Hardware Libraries example program is part of the Altera® SoC Embedded Design Suite (EDS). You can run the sample program on a Cyclone V SoC development kit board.

The example program demonstrates using the Hardware Library to programmatically configure the FPGA and exercise soft IP control from the hard processor system (HPS).

HWLIB Sample Application Overview

The Bare Metal sample application uses the HWLIB API to:

- Programmatically configure the FPGA from the HPS
- Initialize and bring up the Advanced eXtensible Interface (AXI) bridge interfaces between the HPS and the FPGA
- Exercise the FPGA soft IP parallel I/O (PIO) core from the HPS to toggle the development board LEDs

The sample application uses the development kit Golden System Reference Design (GSRD) FPGA configuration. The sample application uses the following files:

- FPGA configuration SRAM Object File (**.sof**)
- Preloader executable file for proper initialization of the GSRD HPS component

The sample application is built with a makefile that performs the following steps:

1. Copies Hardware Libraries source code from installation folder to the current project folder.
2. Compiles the example C source code files with the GNU Compiler Collection (GCC) tool chain from Mentor Graphics®
3. Copies the **.sof** file from the GSRD folder
4. Converts the **.sof** file to a compressed Raw Binary File (**.rbf**) format with the `quartus_cpf` utility available in the Altera Complete Design Suite or the Quartus® II software programmer.
5. Converts the **.rbf** to an equivalent Executable and Linking Format File (**.elf**) object file with the GCC `objcopy` utility.
6. Links the example program and the FPGA configuration resource object files into the HWLIB example executable file.

A debugger script performs the following steps to help execute the sample application:

1. Loads the preloader image, and places a breakpoint at the end of the image
2. Runs the preloader image until it reaches the breakpoint; this properly configures the HPS component according to the GSRD
3. Loads the HWLIB sample application

Related Information

- **Hardware Library Overview** on page 9-1
  For more information, refer to the Hardware Libs Overview section in this document.
- **Mentor Code Sourcery**
  For more information about the Sourcery CodeBench Lite Edition including ARM GCC IDE, refer to the *Embedded Software* page on the Mentor Graphics website.
- **ARM DS-5 Documentation**
  The ARM DS-5 Altera Edition reference material can be accessed online on the documentation page of the ARM website (www.arm.com); and from Eclipse by navigating to **Help > Help Contents > ARM DS-5 Documentation**.

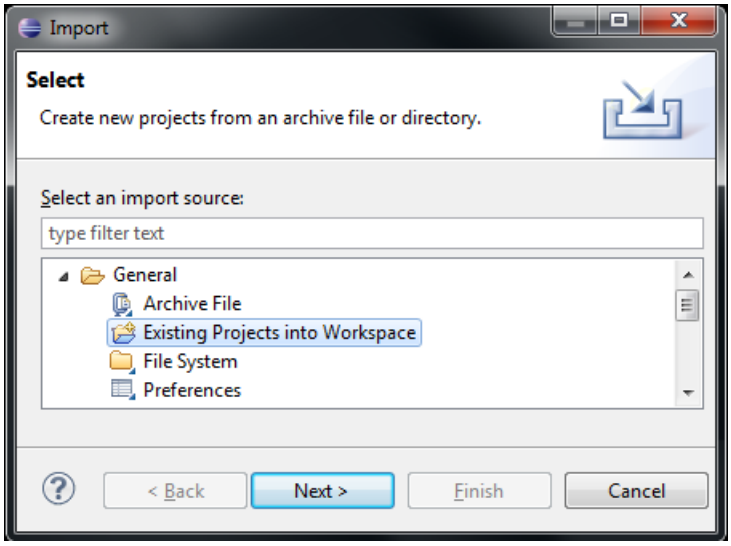
Starting the Eclipse IDE

1. Select **Start Menu > Programs > ARM DS-5 > Eclipse for DS-5** to start Eclipse. Alternatively, you can run `eclipse` command from the **Embedded Command Shell**.
2. The Eclipse tool, part of ARM DS-5 AE, prompts for the workspace folder to be used. Use the suggested folder and click **OK**.
3. The ARM DS-5 AE "Welcome" screen appears. It is instructive, and can be used to access documentation, tutorials and videos.

Importing the Sample Application

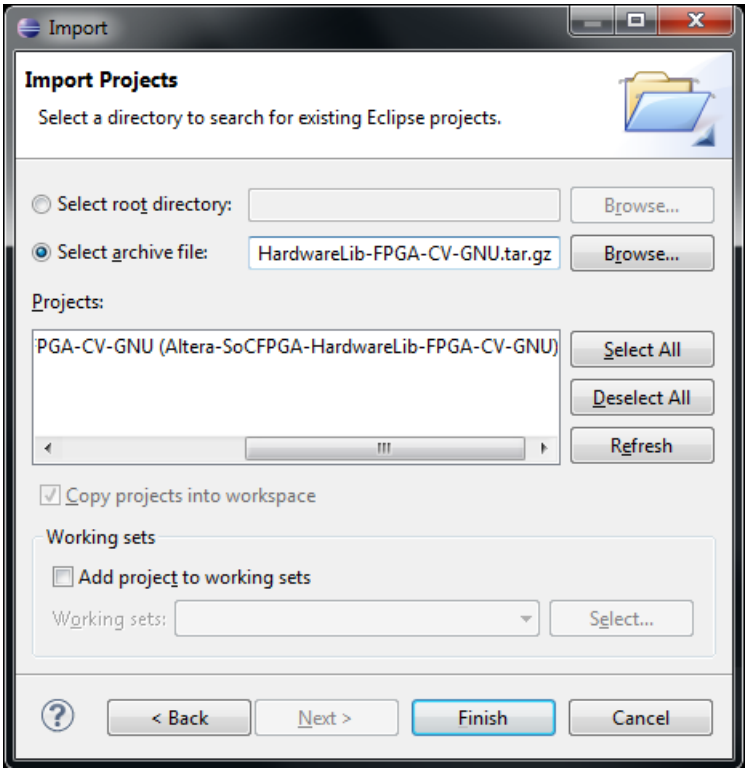
1. In Eclipse, select **File > Import**. The **Import** dialog box displays.
2. In the **Import** dialog box, select **General > Existing Projects into Workspace** and click **Next.** This will open the **Import Projects** dialog box.

Figure 5-10: Import Existing Project



3. In the **Import Projects** dialog box, select the **Select Archive File** option.
4. Click **Browse**, then navigate to **<SoC EDS Location>\embedded\examples\software**, select the file **Altera-SoCFPGA-HardwareLib-FPGA-CV-GNU.tar.gz** and click **OK**.

Figure 5-11: Select Imported File



5. Click **Finish**. The project will be imported. The project files will be displayed in the **Project Explorer** panel. The following files are part of the project:

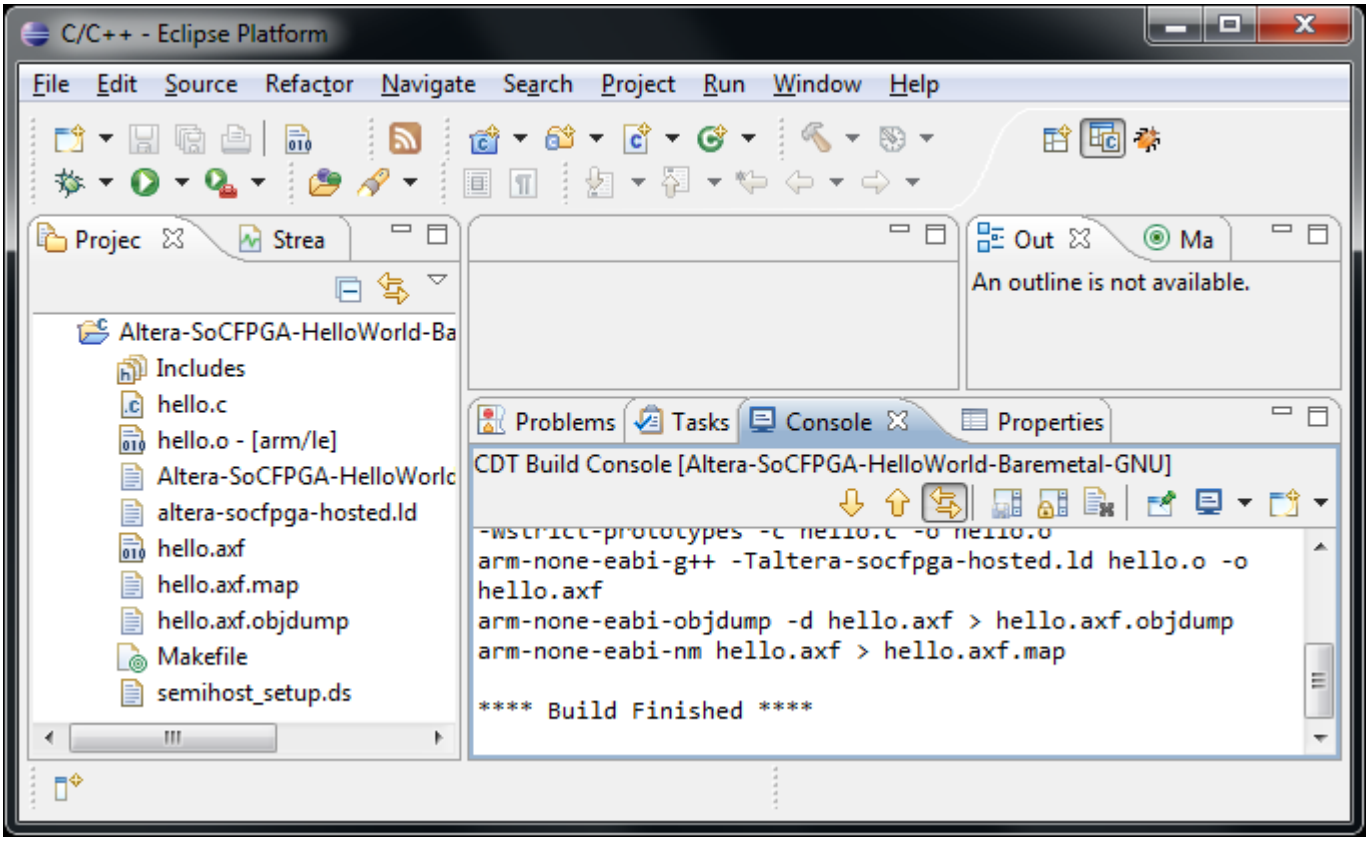
Table 5-2: Project Files

File Name	Description
hwlib.c	Sample application source code
Altera-SoCFPGA-HardwareLib-FPGA-CV-GNU-Debug.launch	Launcher file used to run/debug the sample application from within Eclipse
altera-socfpga-hosted.ld	Linker script
debug-hosted.ds	Debugger script use to load the sample application
Makefile	Makefile used to compile the sample application

Compiling the Sample Application

1. To compile the application, select the project in **Project Explorer**.
2. Select **Project** > **Build Project**.
3. The project compiles and the **Project Explorer** shows the newly created **hwlib.axf**executable file as shown in the above figure. The **Console** dialog box shows the commands that were executed.

Figure 5-12: Project Compiled



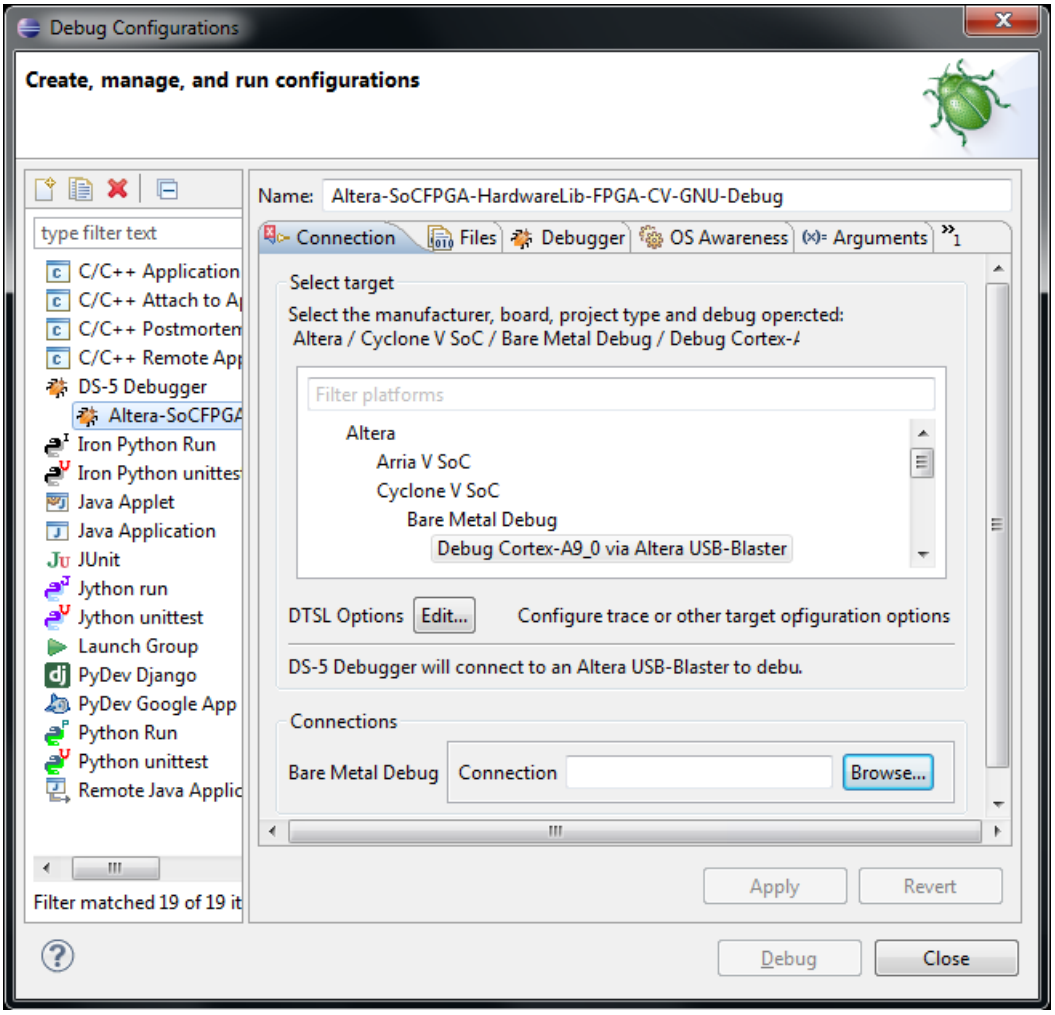
Running the Sample Application

The bare-metal sample application comes with a pre-configured Eclipse **Workspace Launcher** that allows you to load, run, and debug the sample application. The **Workspace Launcher** uses the Altera USB-Blaster II board connection. It uses a debugger script to load and run the Preloader to configure the HPS component, and then loads the sample application

To run the sample application, perform the following steps:

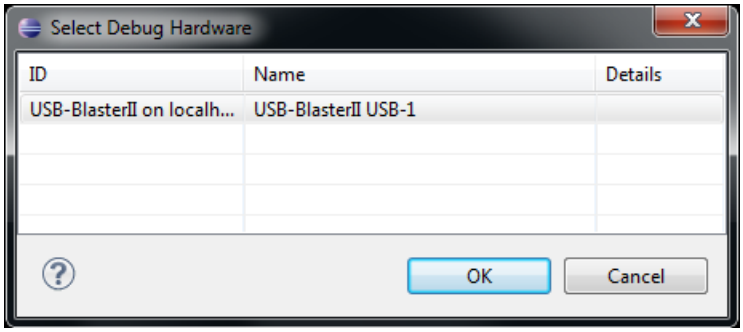
1. In the Eclipse IDE, click **Run > Debug Configurations...** to open the **Debug Configurations** dialog box.
2. In the **Connection** tab in the **Debug Configurations** dialog box, ensure the selected target is **Altera > Cyclone V > Bare Metal Debug > Debug Cortex-A9_0 via Altera USB-Blaster**.
3. Under **Connections** tab, click **Browse** to select the USB Blaster connection.

Figure 5-13: Debug Configurations



4. In the **Select Debug Hardware** dialog box, select the desired USB Blaster and click **OK**.

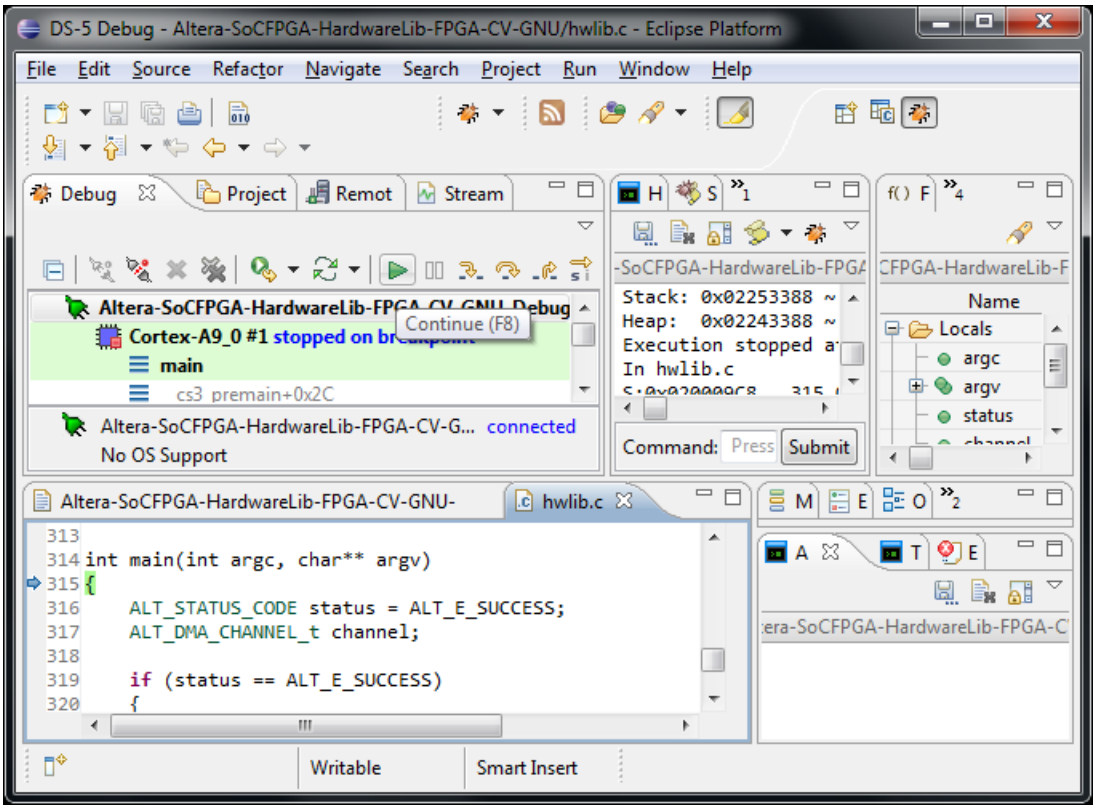
Figure 5-14: Select USB Blaster



5. Click the **Debug** button from the bottom of the **Debug Configurations** dialog box.
6. Eclipse ask whether to switch to **Debug Perspective**. Click **Yes** to accept it.
   The debugger downloads the application on the board through JTAG, enables semi-hosting using the provided script, and runs the application until the PC reaches the **main** function.

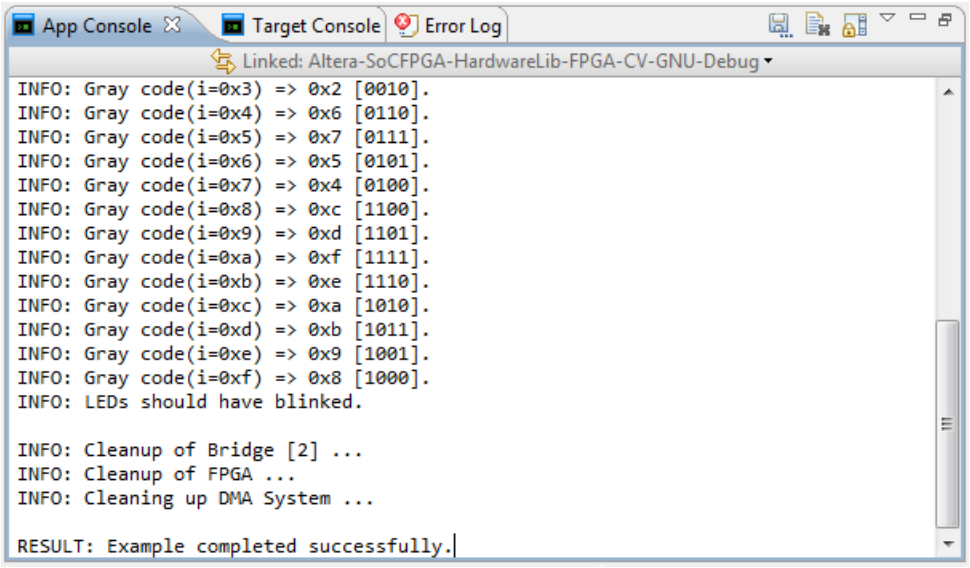
   At this stage, all the debugging features of DS-5 can be used, such as viewing and editing registers and variables, looking at the disassembly code.

Figure 5-15: Application Downloaded



7. Click **Continue** green button (or press **F8**) to run the application. It displays a log of activities it performs in the **Application Console**.

Figure 5-16: Application Completed


App Console
Target Console
Error Log
Linked: Altera-SoCFPGA-HardwareLib-FPGA-CV-GNU-Debug
INFO: Gray code(i=0x3) => 0x2 [0010].
INFO: Gray code(i=0x4) => 0x6 [0110].
INFO: Gray code(i=0x5) => 0x7 [0111].
INFO: Gray code(i=0x6) => 0x5 [0101].
INFO: Gray code(i=0x7) => 0x4 [0100].
INFO: Gray code(i=0x8) => 0xc [1100].
INFO: Gray code(i=0x9) => 0xd [1101].
INFO: Gray code(i=0xa) => 0xf [1111].
INFO: Gray code(i=0xb) => 0xe [1110].
INFO: Gray code(i=0xc) => 0xa [1010].
INFO: Gray code(i=0xd) => 0xb [1011].
INFO: Gray code(i=0xe) => 0x9 [1001].
INFO: Gray code(i=0xf) => 0x8 [1000].
INFO: LEDs should have blinked.
INFO: Cleanup of Bridge [2] ...
INFO: Cleanup of FPGA ...
INFO: Cleaning up DMA System ...
RESULT: Example completed successfully.

8. Click **Disconnect from Target** button to close the debugging session.

Sequence	Sample Application Function	Used Hardware Libraries APIs	Description
1	socfpga_dma_setup	alt_dma_init	Init DMA module driver
2		alt_dma_channel_alloc_any	Allocate DMA channel
3		alt_dma_channel_state_get	Check state of DMA channel
4	socfpga_fpga_setup	alt_fpga_init	Init FPGA manager driver
5		alt_fpga_state_get	Query the FPGA state
6		alt_fpga_control_enable	Enable controlling the FPGA
7		alt_fpga_cfg_mode_get	Query the configuration mode
8		alt_fpga_configure_dma	Configure the FPGA using the DMA
9	socfpga_bridge_setup	alt_bridge_init	Initialize bridges
10		alt_addr_space_remap	Remap address space
11	socfpga_bridge_io	N/A	Application accesses Soft IP directly
12	socfpga_bridge_cleanup	alt_bridge_uninit	Deinitialize bridges

Sequence	Sample Application Function	Used Hardware Libraries APIs	Description
13	socfpga_fpga_cleanup	alt_fpga_control_ disable	Disable control of FPGA
14		alt_fpga_uninit	Close the FPGA driver
15	socfpga_dma_cleanup	alt_dma_channel_free	Deallocate the DMA channel
16		alt_dma_uninit	Close the DMA driver

Getting Started with Peripheral Register Visibility

The ARM DS-5 Altera Edition allows you to specify the peripheral IP register descriptions using **.svd** files. The **.svd** files are resulted from the hardware project compilation using ACDS.

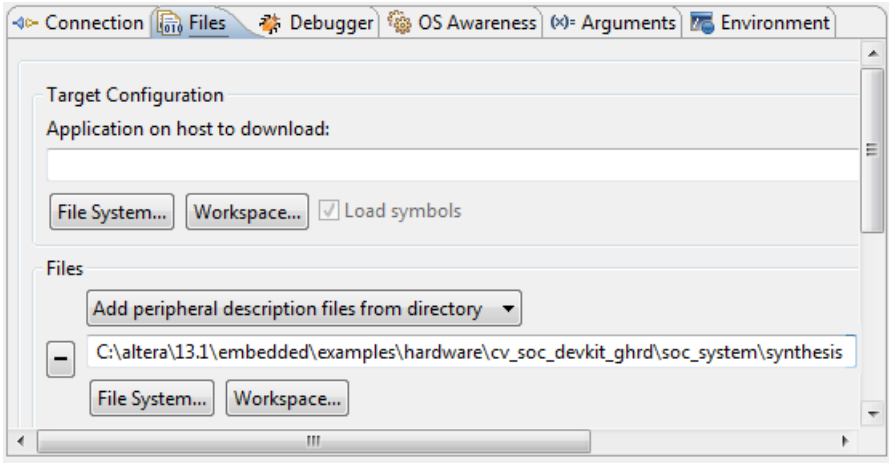
The **.svd** files contain the description of both HPS peripheral registers, such as UART, EMAC, and timers; and the Soft IP peripheral registers residing on FPGA side.

This section presents the necessary steps in order to view the HPS registers and the Soft IP registers using the *Getting Started with Hardware Library* example.

Note: The soft IP register descriptions are not generated for all soft IP cores. Do not expect to have registers for all the cores they use on FPGA. Some may have it, some may not.

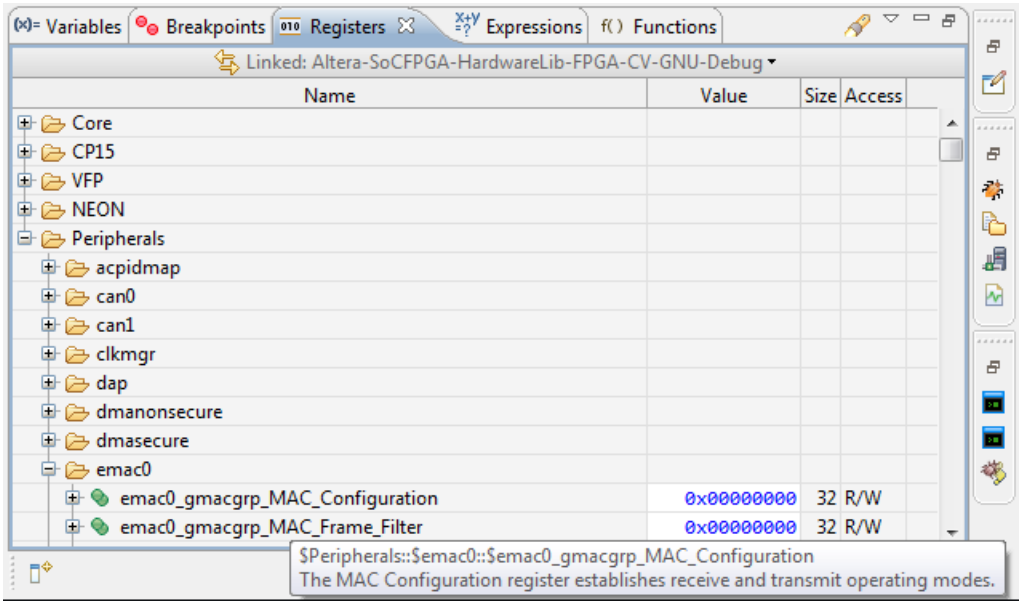
1. Perform the steps described in the Getting Started with Hardware Library section up to and including configuring the USB Blaster connection.
2. In the Eclipse IDE, click **Run > Debug Configurations...** to open the **Debug Configurations** dialog box.
3. In the **Debug Configurations** dialog box, go to the **Files** panel and under the **Files** panel:
   a. Select **Add peripheral description files from directory** from the drop down box
   b. Use the browse File System button to browse to the folder **<SoC EDS Folder>\examples\hardware\cv_soc_devkit_ghrd\soc_system\synthesis.** This is where the **.svd** file generated by Quartus II is located.

Figure 5-17: Configue Peripheral Register Visibility



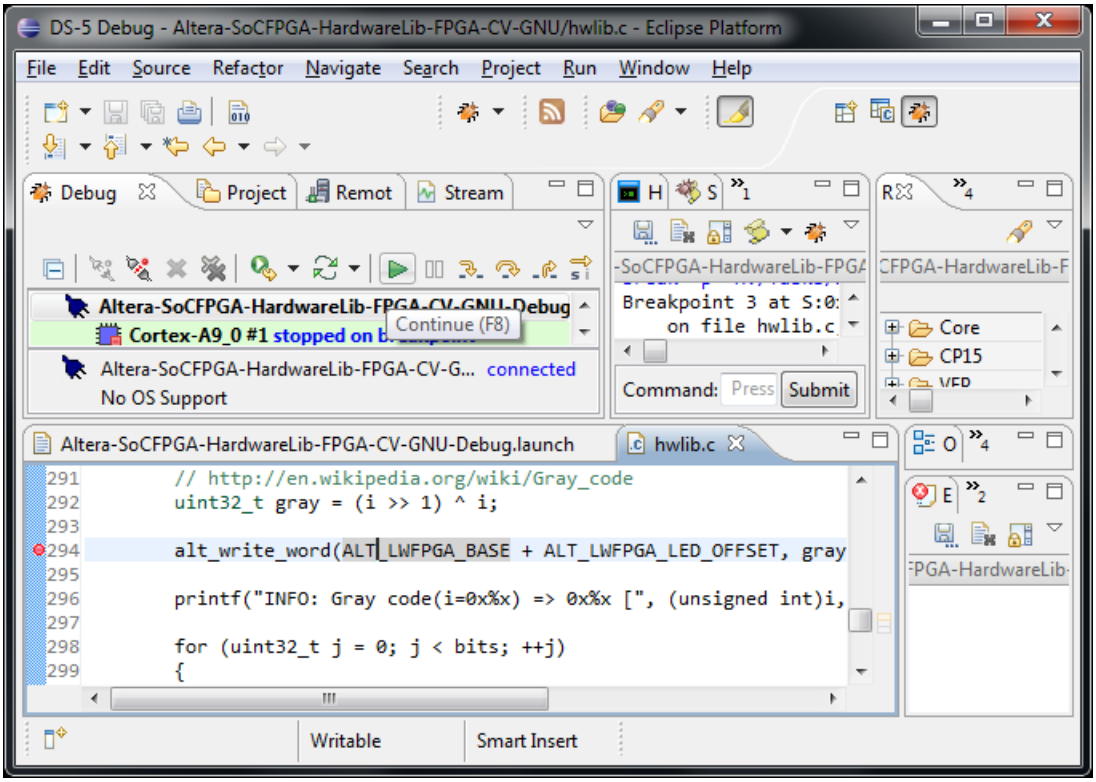
4. Click the **Debug** button to download the application to the target board.
5. Select the **Registers** view and maximize it. It shows the Core, Coprocessor, VFP, NEON and Peripheral Registers. Under the Peripherals group, the DS-5 displays both the HPS peripheral registers and the Soft IP registers. The figure below shows some of the HPS modules, with the EMAC one expanded.

Figure 5-18: Peripheral Registers



6. Put a breakpoint in the source code file named **hwlib.c** at the line where the soft IP GPIO module data register is written to turn LEDs ON or OFF. The breakpoint is added by simply double-clicking to the left of the line number in the dialog box.

Figure 5-19: Breakpoint Added

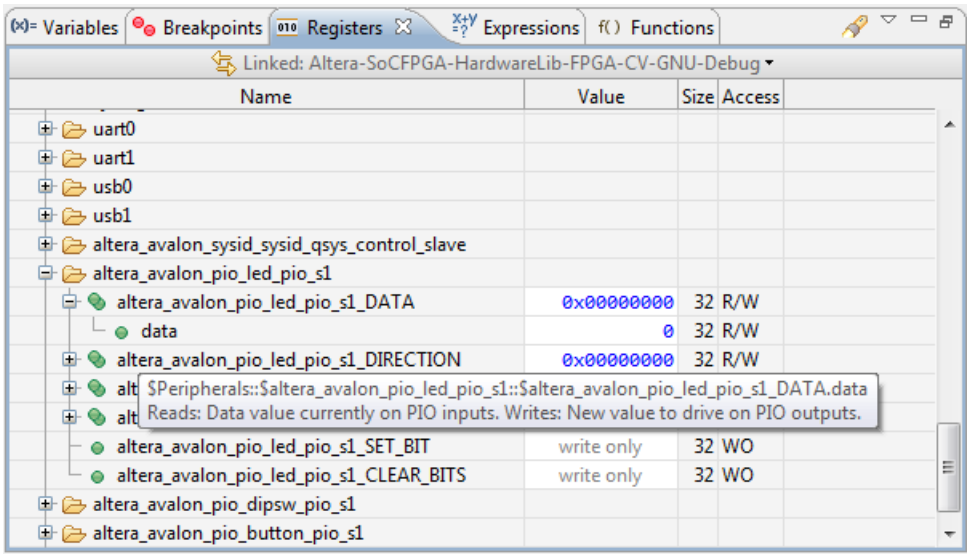


7. Let the program run by clicking the green **Continue** button or by pressing F8. The code will stop at the breakpoint.

   Note: This ensures that when you try to access the soft IP registers, they are already available. If you try to access the soft IP registers before the FPGA is programmed or before the bridges are open, the debugger generates a memory access abort and the debugging session fails.

8. Maximize the **Registers** dialog box and expand the **Peripherals register** group
9. Scroll to the end of the list and expand the **altera_avalon_pio_led_pio_s1** group. It corresponds to the soft IP GPIO module that controls the FPGA LEDs on the board.
10. Expand the DATA register. This register contains the values that are driven on the GPIO pins to control the LEDs.

Figure 5-20: Soft IP Registers



11. You can resume the code several times by pressing F8, and you will see how the **DATA** register changes and the HPS LEDs on the board are lighted accordingly.
12. You can also change the **DATA** register, manually and see the LEDs being lighted accordingly.
13. Collapse the soft IP register group to avoid the debugger accessing them on the next debugging session before they are accessible.
14. Click **Disconnect from Target** button to close the debugging session.

Note: Do not try to access the soft IP registers before the FPGA is programmed or before the bridges are open. Otherwise, the debugger will generate a memory access abort and the debugging session will fail. This includes having any soft IP registers groups expanded in the **Registers** dialog box. The debugger will try to access them in order to refresh the view and it will generate a memory access abort if they are not accessible. Always collapse the soft IP register view after usage if there is any chance they will not be available to the debugger.

Related Information

- **Getting Started with the Hardware Library** on page 5-11
  For more information, refer to the *Getting Started with the Hardware Library* section.
- **ARM DS-5 Documentation**
  The ARM DS-5 Altera Edition reference material can be accessed online on the documentation page of the ARM website (www.arm.com); and from Eclipse by navigating to **Help > Help Contents > ARM DS-5 Documentation**.

Getting Started with Linux Application Debugging

The ARM DS-5 Altera Edition provides very powerful Linux application debugging capabilities.

This section presents running the ARM DS-5 Altera Edition for the first time, importing, compiling and running the Hello World Linux example application provided as part of SoC EDS.

Note: This section uses a Linux host computer, as can be seen from the screen shots and the issued commands. However, the scenario can also be run on a Windows machine, although it is not usual for Linux development to be done on Windows.

Related Information

- **ARM DS-5 Altera Edition** on page 6-1
  For more information, refer to the *ARM DS-5 Altera Edition* section.
- **ARM DS-5 Documentation**
  The ARM DS-5 Altera Edition reference material can be accessed online on the documentation page of the ARM website (www.arm.com); and from Eclipse by navigating to **Help > Help Contents > ARM DS-5 Documentation**.
- **Rocket Boards**
  For more information about Linux, refer to the Rocketboards website.

Configuring Linux

For this getting started scenario we need Linux to be running on the target board and be connected to the local network. The local network has to have a DHCP server that will allocate an IP address to the board.

Eclipse needs an account with a password to be able to connect to the target board. The root account does not have a password by default, so one needs to be set up.

The required steps are:

1. Setup the board as described in the *Getting Started with Board Setup* section; and connect the HPS Ethernet Connector J2 to the local network.
2. Start Linux on the target board, as described in the *Getting Started with Running Linux* section.
3. On the Linux console, run the command `ifconfig` to determine the IP address of the board.
4. On the Linux console, change the root password by running the `passwd` command. Ignore the warnings about a weak password.

Related Information

- **Getting Started with Board Setup** on page 5-1
  For more information, refer to the *Getting Started with Board Setup* section.
- **Getting Started with Running Linux** on page 5-2
  For more information, refer to the *Getting Started with Running Linux* section.

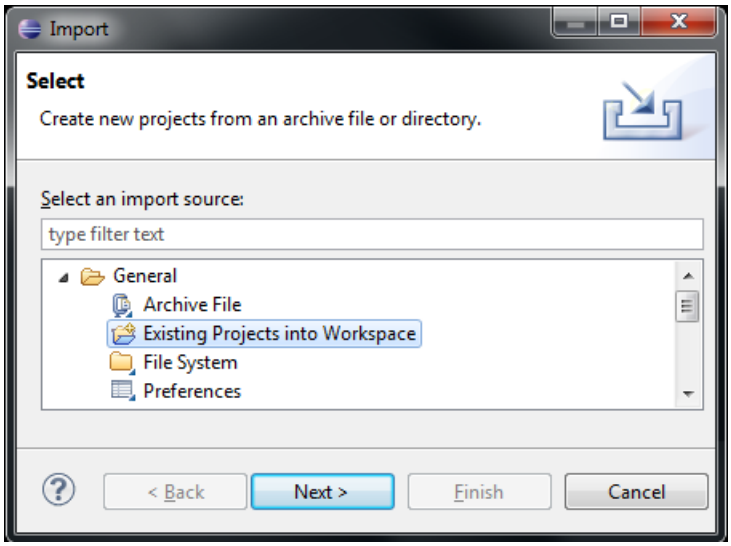
Starting Eclipse via the Embedded Command Shell

1. Start an **Embedded Command Shell** by running <*SoC EDS installation directory*>**/embedded_command_shell.sh**.
2. Start Eclipse by running the `eclipse` command from the **Embedded Command Shell**.
3. The Eclipse tool, part of the ARM DS-5 AE, prompts for the workspace folder to be used. Accept the suggested folder and click **OK**.
4. The ARM DS-5 AE "Welcome" screen appears. It can be used to access documentation, tutorials, and videos.

Importing the Sample Application

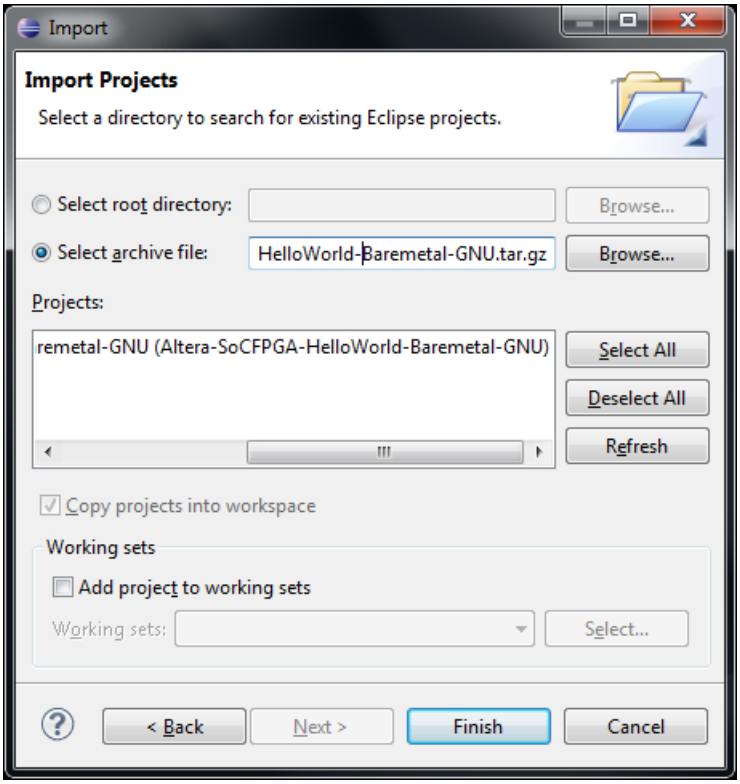
1. In Eclipse, select **File > Import**. The **Import** dialog box displays.
2. In the **Import** dialog box, select **General > Existing Projects into Workspace** and click **Next.** This will open the **Import Projects** dialog box.

Figure 5-21: Import Existing Project



3. In the **Import Projects** dialog box, select the **Select Archive File** option.
4. Click **Browse**, then navigate to <*SoC EDS installation directory*>**\embedded\examples\software**, select the file **Altera-SoCFPGA-HelloWorld-Baremetal-GNU.tar.gz** and click **OK**.

Figure 5-22: Select Imported File



5. Click **Finish**. The project is imported. The project files are displayed in the **Project Explorer** panel. The following files are part of the project:

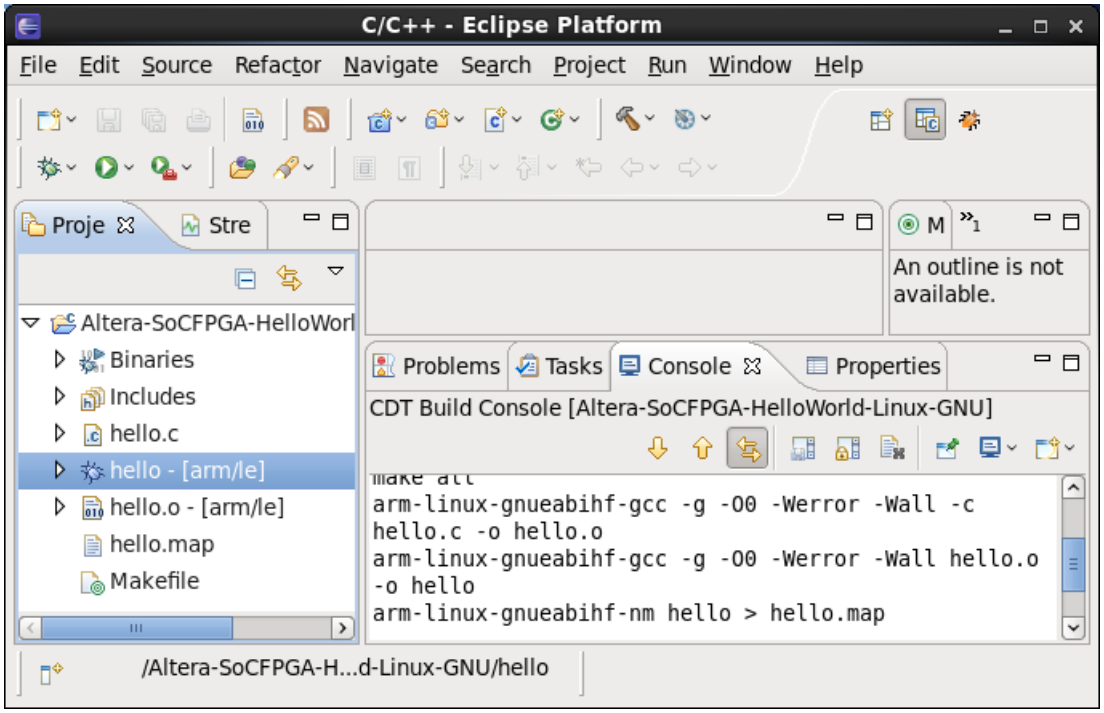
Table 5-3: Project Files

File Name	Description
hello.c	Sample application source code
Makefile	Makefile used to compile the sample application

Compiling the Sample Application

1. To compile the application, select the project in **Project Explorer**.
2. Select **Project > Build Project**.
3. The project compiles and the **Project Explorer** shows the newly created **hello**executable file as shown in the figure below. The **Console** dialog box shows the commands that were executed.

Figure 5-23: Project Compiled



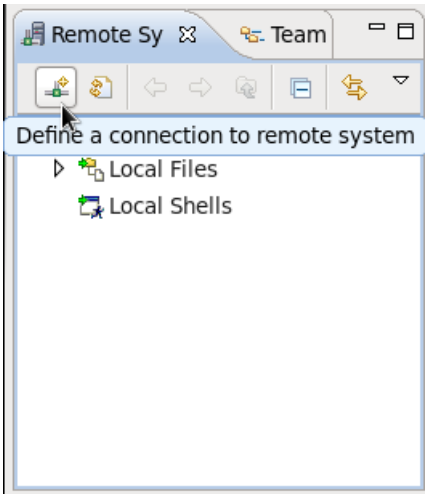
Setting up Remote System Explorer

The ARM DS-5 AE can run and debug programs directly on the target with the help of the Remote System Explorer (RSE). Before this feature can be used, the RSE needs to be configured to connect to the target board running Linux.

1. In your Eclipse workspace, select **Window > Open Perspective > Other**. This will open the **Open Perspective** dialog box.
2. In the **Open Perspective** dialog box, click the **Remote System Explorer** and click **OK**.
3. In the Remote System Explorer view, right click **Local** and select **New > Connection ...**.This will open the **New Connection** wizard.

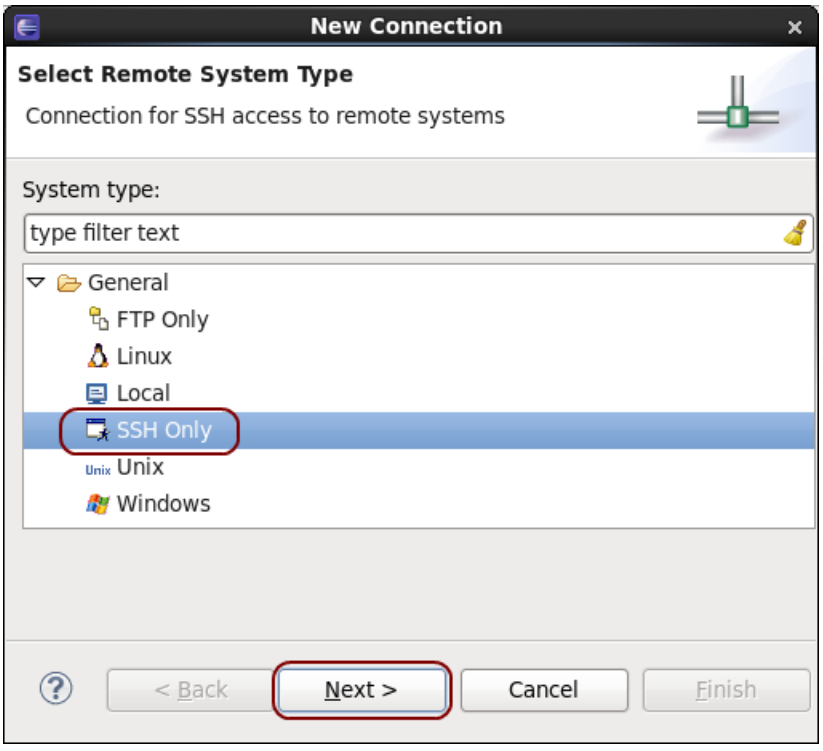
   Note: Clicking the + sign achieves the same result.

Figure 5-24: Create New Connection



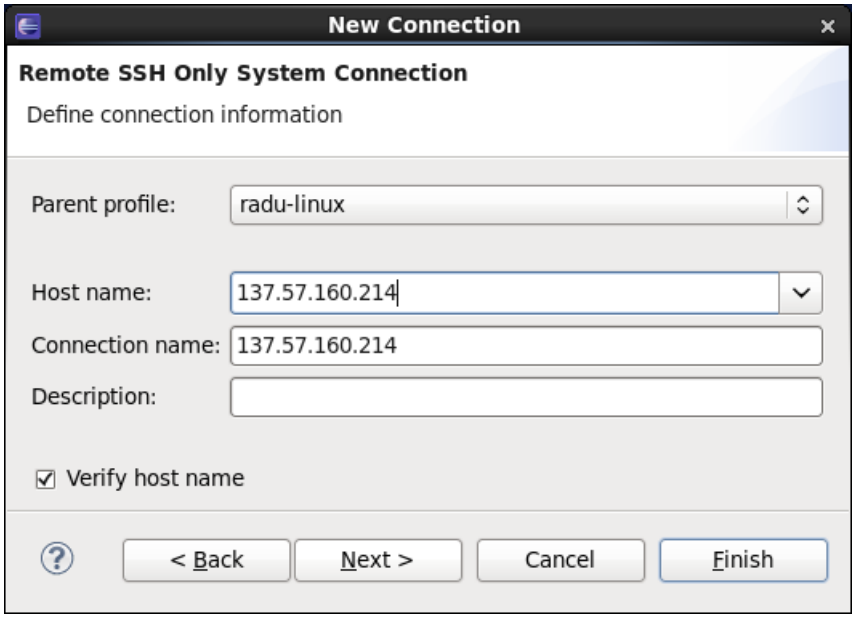
4. In the first page of the **New Connection** wizard, named Remote System Type view, select **SSH only** and click **Next**.

Figure 5-25: New Connection window



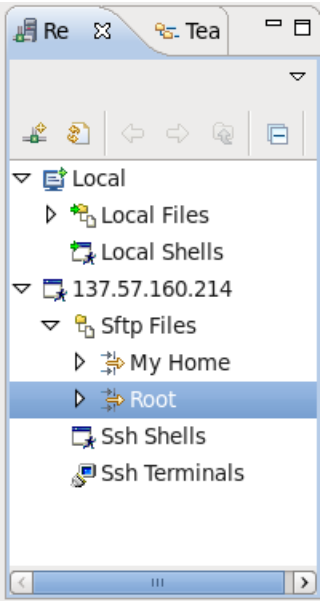
5. Enter the IP address of the board in the **Host Name** field. Click **Finish** to create the connection.

Figure 5-26: Enter Target IP Address



6. In the **Remote Systems** panel, click the **Target IP** > **Sftp Files** > **Root**. This opens a dialog box to enter the username and password.

Figure 5-27: Browse Target

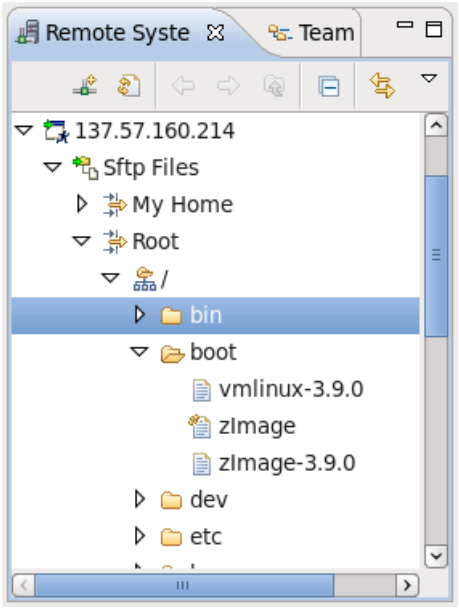


7. Enter **root** as the username and the password you have selected in the *Configuring Linux* section. Check the Save User ID and Save password check boxes. Click **OK**.

Figure 5-28: Target Username and Password

8. Eclipse asks for confirmation of authenticity of the board. Click **Yes**.
9. **Remote System** Explorer shows the files on the DevKit board on the left panel.

Figure 5-29: Target Files



Running Sample Application

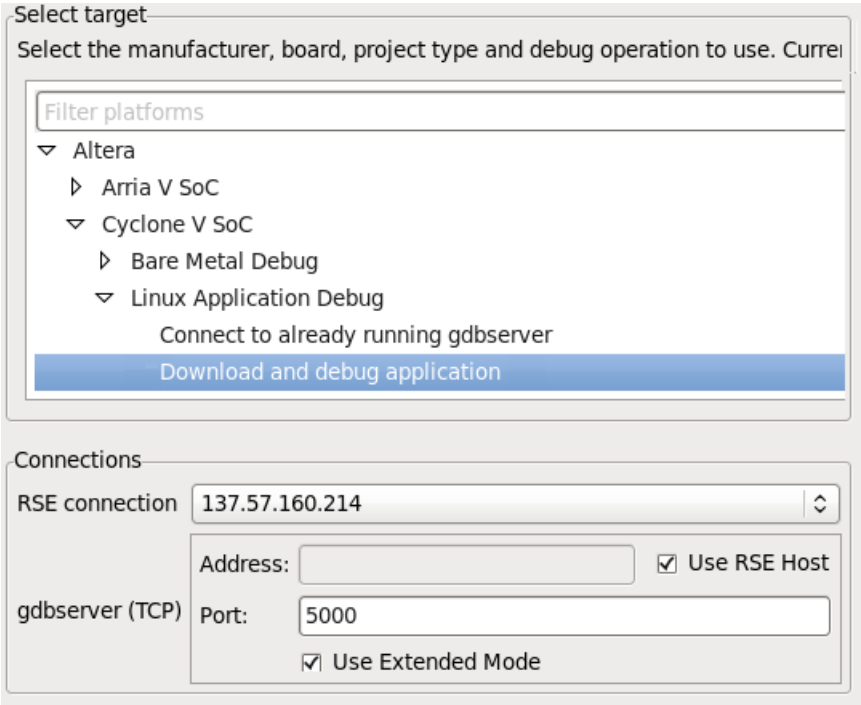
At this stage, we have a compiled Linux application and a properly configured **Remote Systems** Connection. This section shows how to create a Debugger Configuration and use it to run and debug the application.

1. Select **Run > Debug Configurations…** to open the **Debug Configurations** dialog box.
2. Right click the **DS-5 Debugger** and click **New** to create a new debug configuration.
3. Name the newly created debugger configuration, **LinuxAppDebug_DevKit**, by editing its name in the **Connection** tab.
4. In the **Connection** tab, select:

a. For the Free Web Edition license, select **Generic > gdb server > Linux Application Debug > Download and Debug Application**.
b. For the Subscription Edition or 30-day Evaluation Edition, select **Altera > Cyclone 5 SoC > Linux Application Debug > Download and Debug Application**.

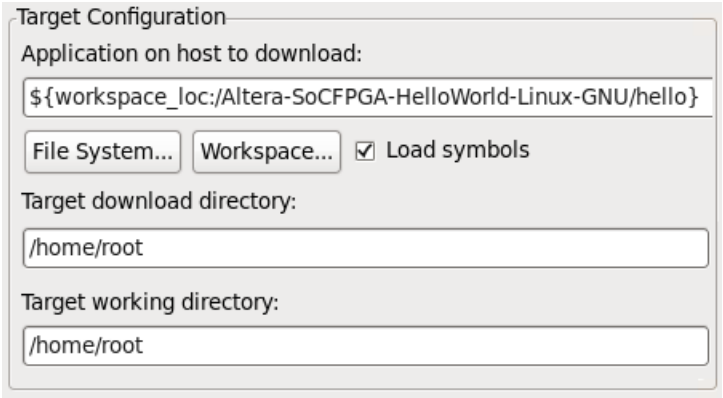
5. In the **Connection** tab, select the newly created RSE connection and keep the default values.

Figure 5-30: Connection Settings



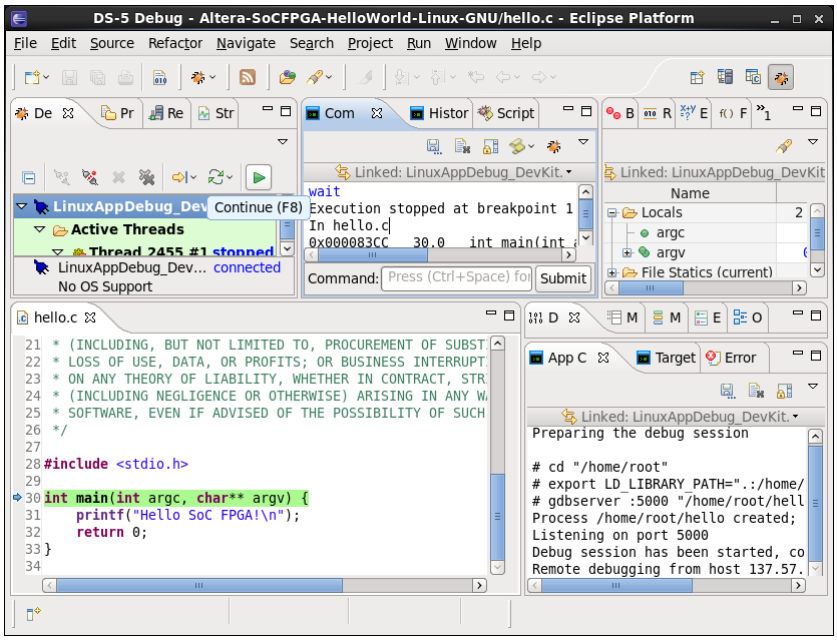
6. Go to **Files** tab, and set the **Target Configuration** parameters:
   a. Select the **Application on host to download** to be the **hello** executable file. Use the **Workspace…** browse button.
   b. Edit the Target **download** directory to be **"/home/root"** (the root folder).
   c. Edit the Target **working** directory to be **"/home/root"** (the root folder).

Figure 5-31: Target Configuration



7. Click the **Debug** button. A dialog window appears asking to switch to **Debug** perspective. Click **Yes**.
8. Eclipse downloads the application to the board and stops at **main** function entry.

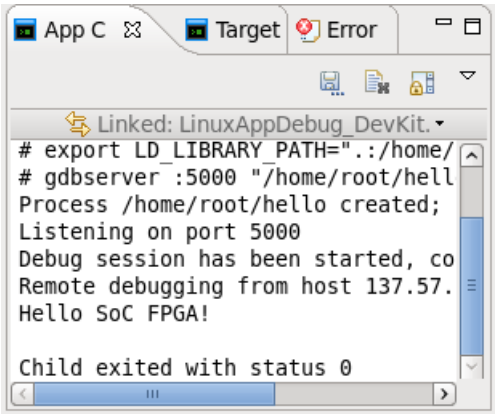
Figure 5-32: Program Downloaded



Note: At this stage, all the usual debugging features of DS-5 can be used, such as breakpoints, view variables, registers, tracing, and threads.

9. Click the **Continue** green button or press **F8** to run the application. The hello message is printed on the **Application Console**.

Figure 5-33: Hello Message



Getting Started with Linux Kernel and Driver Debugging

The ARM DS-5 Altera Edition provides very powerful Linux Kernel and Driver debugging capabilities.

This section presents an example on of how to debug the Linux kernel and drivers using DS-5. The software engineers can use the dedicated Linux debugging features presented in this section together with the basic debugging features such as viewing registers, inspecting variables and setting breakpoints.

Note: In the scenario presented here the Linux kernel is already running on the board, but it can also be downloaded through the debugger.

Note: This scenario uses the pre-built Linux images and Linux source code included in the SoC EDS. These are examples only; use the latest sources from the Rocketboards website (www.rocketboards.org) for development.

Note: This section uses a Linux host computer, as can be seen from the screenshots and the issued commands. However, the scenario can also be run on a Windows machine, although it is not usual for Linux development to be done on Windows.

Note: The paths presented in this section assume the default installation paths were used. Adjust accordingly if non-standard location is used.

Related Information

- **ARM DS-5 Altera Edition** on page 6-1
  For more information, refer to the *ARM DS-5 Altera Edition* section.
- **ARM DS-5 Documentation**
  The ARM DS-5 Altera Edition reference material can be accessed online on the documentation page of the ARM website (www.arm.com); and from Eclipse by navigating to **Help > Help Contents > ARM DS-5 Documentation**.
- **Rocket Boards**
  For more information about Linux, refer to the Rocketboards website.

Prerequisites

1. Make sure the desired Linux kernel version is already running on the board. See the *Getting Started with Running Linux* section for instructions on how to rn the provided Linux binaries on the board.
2. Make sure the Linux kernel executable file is accessible on the host computer. The kernel executable for the pre-built Linux image is located at <*SoC EDS installation directory*>**/embeddedsw/socfpga/prebuilt_images/vmlinux**.
3. Make sure the source code corresponding to the kernel running on the board are accessible on the host computer. The sources for the pre-built Linux image can be obtained by:
   a. Start Embedded Command Shell by running <*SoC EDS installation directory*>**/embedded_command_shell.sh**
   b. Run the following command: cd <*SoC EDS installation directory*>**/embeddedsw/socfpga/sources/**
   c. Run the following command: `./git_clone.sh`

   The above commands will fetch the Linux kernel source code from the Git server on the Rocketboards website (rocketboards.org).

   The source code will be available in the folder <*SoC EDS installation directory*>**/embeddedsw/socfpga/sources/linux-socfpga**.

   Note: if your computer connects to the Internet using a proxy, you will need to use this command to tell the **Git** utility about the proxy: `git config --global http.proxy <proxy_name>`

Related Information

- **Getting Started with Running Linux** on page 5-2
  For more information, refer to the Getting Started with Running Linux section in this document.
- **Rocket Boards**
  For more information about Linux and the latest source releases, refer to the Rocketboards website.

Starting Eclipse via the Embedded Command Shell

1. Start an **Embedded Command Shell** by running <*SoC EDS installation directory*>**/embedded_command_shell.sh**.
2. Start Eclipse by running the `eclipse` command from the **Embedded Command Shell**.
3. The Eclipse tool, part of the ARM DS-5 AE, prompts for the workspace folder to be used. Accept the suggested folder and click **OK**.
4. The ARM DS-5 AE "Welcome" screen appears. It can be used to access documentation, tutorials, and videos.

Debugging the Kernel

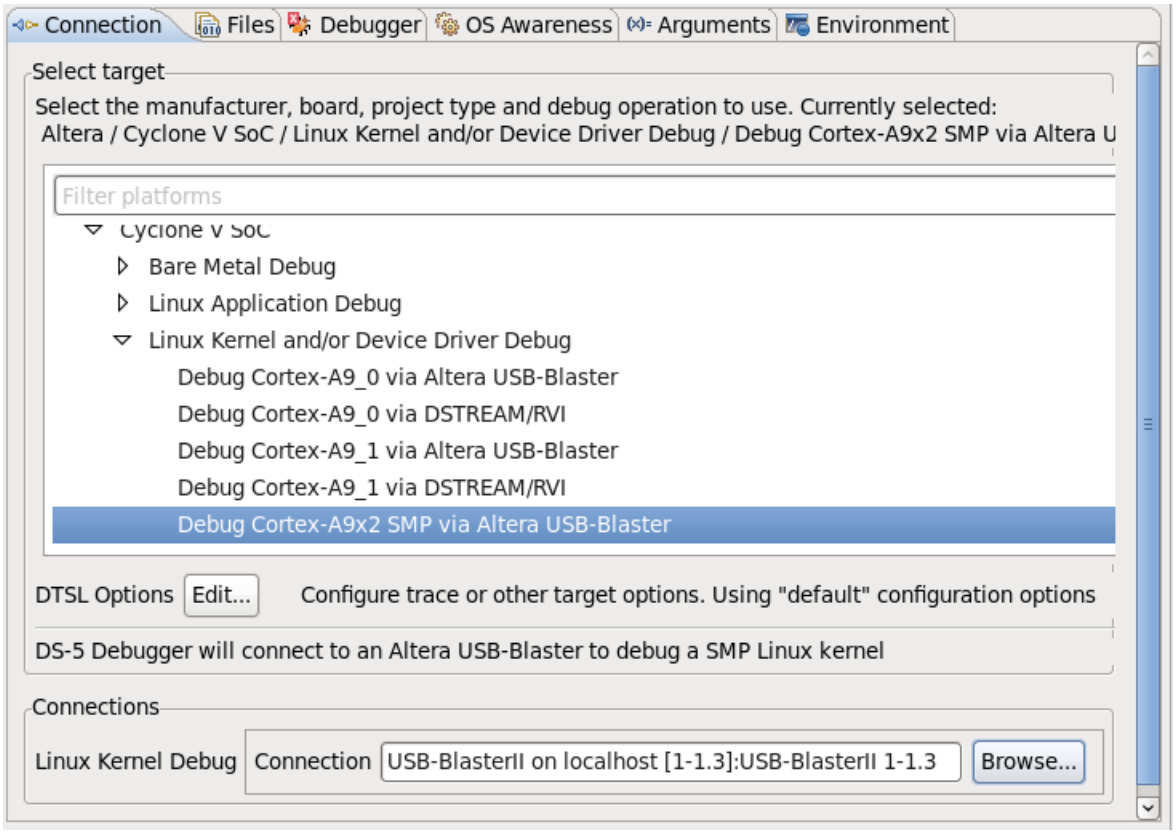
This section presents how to create a Debug Configuration that is then used to debug the Linux kernel.

1. Select **Run > Debug Configurations**… to open the **Debug Configurations** dialog box.
2. In the **Debug Configurations** dialog box, right-click **DS-5 Debugger** on the left panel and select **New**.
3. In the **Debug Configurations** dialog box, perform the following:
   a. Rename the configuration to **DebugLinux_DevKit** using the **Name** edit box

b. Select the Target to be **Altera > CycloneVSoC >Linux Kernel and/or Device Driver Debug >Debug Cortex-A9x2 SMP via Altera USB-Blaster**

c. Click the **Browse** button near the Connection edit box and select the desired USB Blaster instance

Figure 5-34: Configure Connection


Connection
Files
Debugger
OS Awareness
Arguments
Environment
Select target
Select the manufacturer, board, project type and debug operation to use. Currently selected:
Altera / Cyclone V SoC / Linux Kernel and/or Device Driver Debug / Debug Cortex-A9x2 SMP via Altera U
Filter platforms
Cyclone V SoC
Bare Metal Debug
Linux Application Debug
Linux Kernel and/or Device Driver Debug
Debug Cortex-A9_0 via Altera USB-Blaster
Debug Cortex-A9_0 via DSTREAM/RVI
Debug Cortex-A9_1 via Altera USB-Blaster
Debug Cortex-A9_1 via DSTREAM/RVI
Debug Cortex-A9x2 SMP via Altera USB-Blaster
DTSL Options
Edit...
Configure trace or other target options. Using "default" configuration options
DS-5 Debugger will connect to an Altera USB-Blaster to debug a SMP Linux kernel
Connections
Linux Kernel Debug
Connection
USB-BlasterII on localhost [1-1.3]:USB-BlasterII 1-1.3
Browse...

4. Click the **Files** tab.

a. Specify the Linux kernel executable

b. Specify the symbols file

Note: This tab also shows the soft IP peripheral registers, as described in ARM DS-5 AE - soft IP Register Visibility section

5. Click on the **Debugger** and perform the following steps:

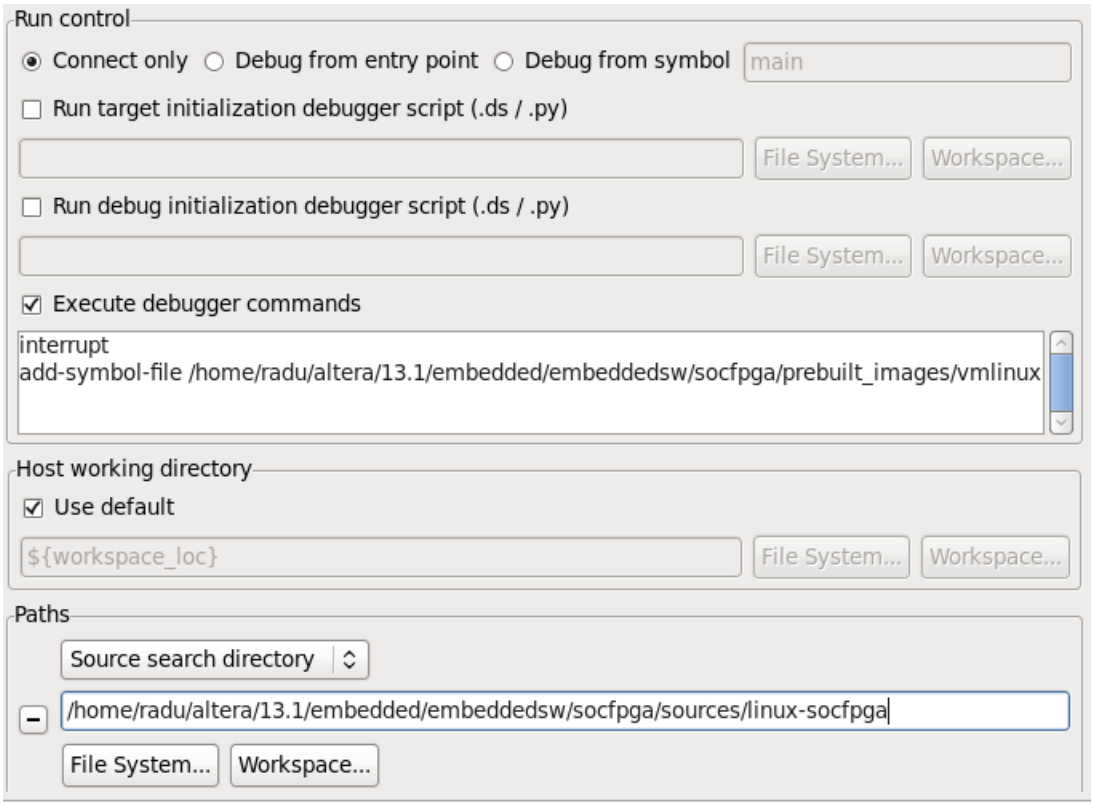
a. Select option **Connect Only** for **Run Control**

b. Check **Execute debugger** commands check box

c. Add the `debugger` commands to stop cores and load image symbols for the Linux executable, as shown in the following figure

d. Add the path to the Linux source files on the host machine to allow the debugger to locate them

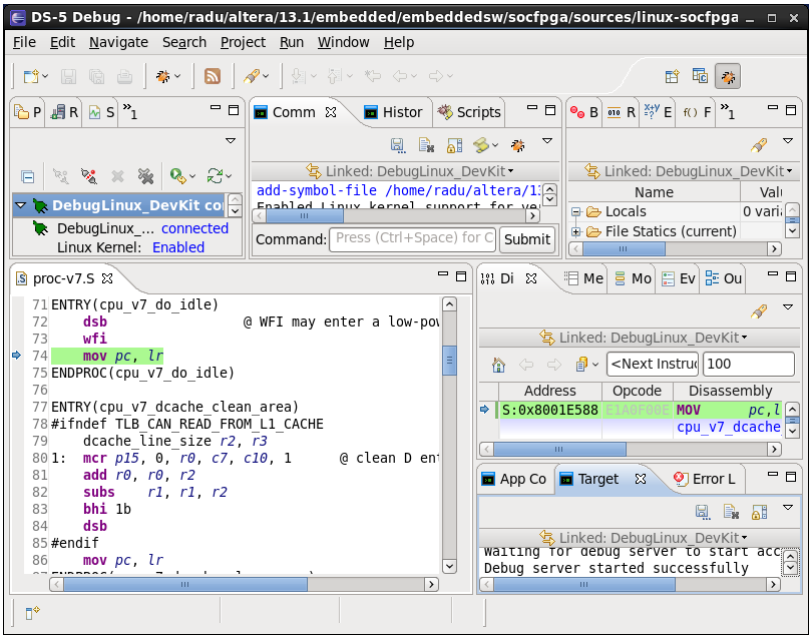
Figure 5-35: Debugger Settings



6. Click the **Debug** button. The debugger connects to the board, stops the cores as instructed and loads the kernel symbols. It determines where the cores are stopped, and highlights it in the source code.

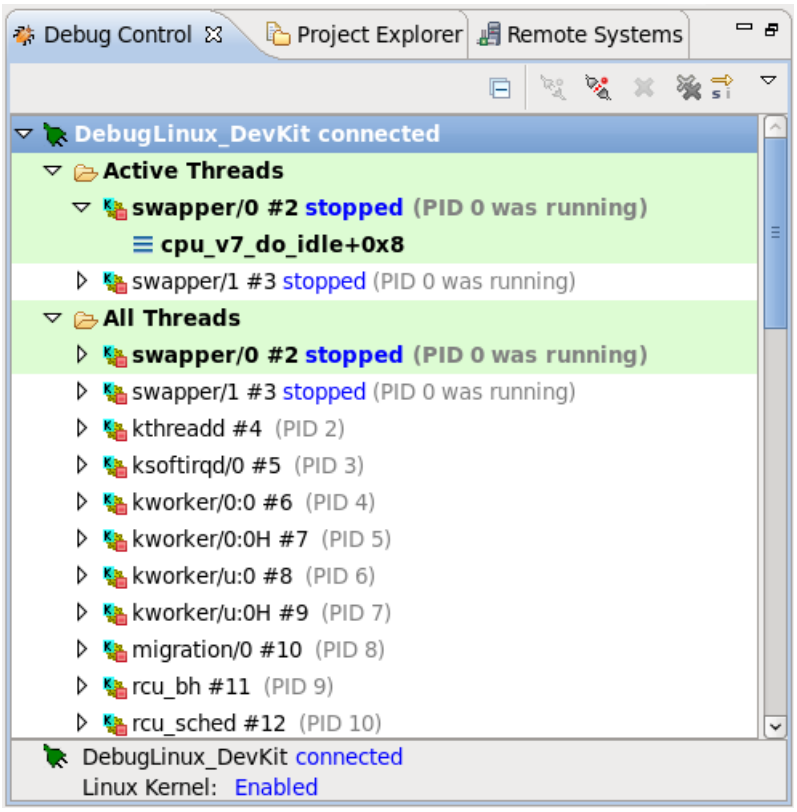
   The following figure shows the debugger stopped in the idle instruction.

Figure 5-36: Linux Kernel Stopped



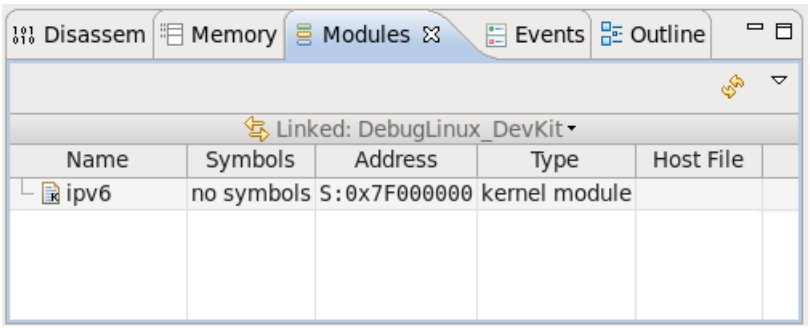
7. To view the running threads, maximize the top left panel. It shows **Active Threads** with the two currently executing threads. Also the **All Threads** can be expanded to show all threads in the system.

Figure 5-37: Linux Threads



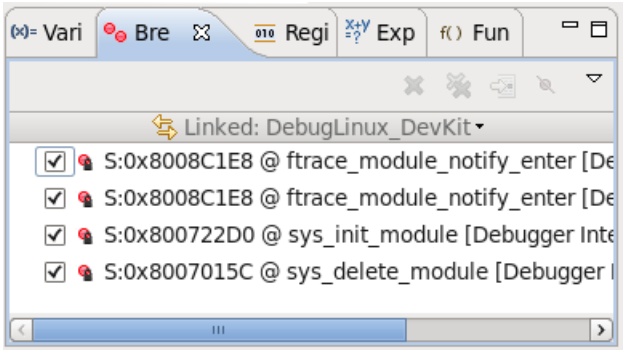
8. Minimize the **Debug Control** panel and maximize the **Functions** panel from top right. All of the functions in the kernel are displayed. The **Functions** panel supports the following operations for each function:
   a. Run up to the function
   b. Set PC to function
   c. Locate in source code/memory/disassembly
   d. Set breakpoints software/hardware
   e. Set trace points: enable/disable/toggle
9. Select **Modules** panel to view the currently loaded modules. In the example below only the **ipv6** module is loaded.

Figure 5-38: Linux Kernel Modules



10. Add breakpoints at the `module load` and `module unload` functions. As modules are loaded with `insmod`, and removed with `rmmod`, the DS-5 AE will reflect the changes.

Figure 5-39: Kernel Debugger Breakpoints



Getting Started with Tracing

ARM DS-5 provides powerful tracing features, allowing PTM and STM tracing. It allows different tracing data destination types.

This section presents an example of Program Tracing using PTM and storing the tracing information in memory using ETF.

The tracing scenario presented here uses Linux kernel debugging as an example, but any application can be traced in the same way.

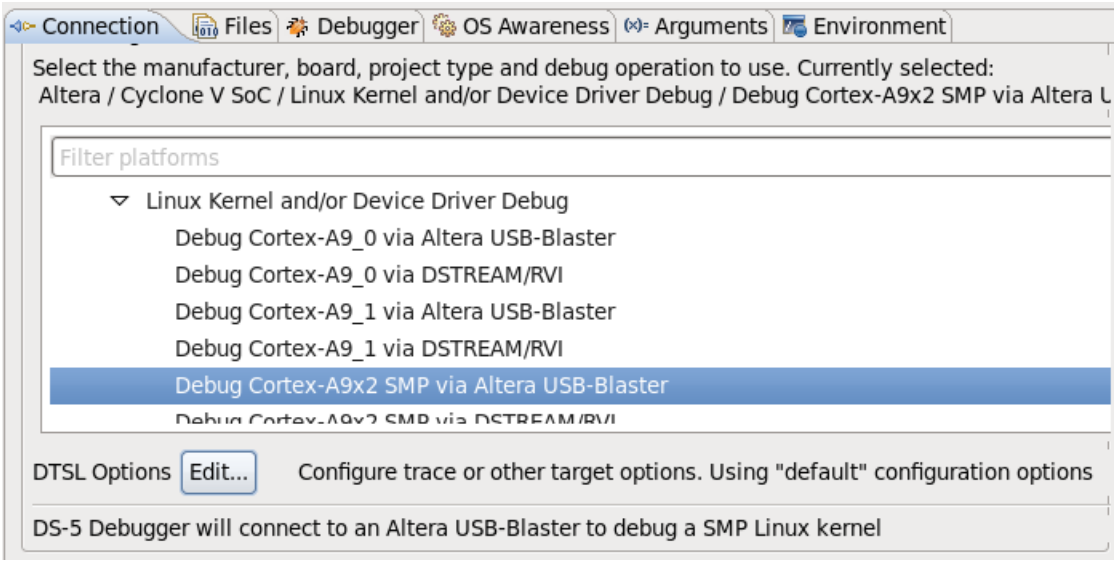
As shown, the tracing can be selected to show current core, a particular core, or follow the currently executing thread.

The following steps are necessary in order to enable PTM tracing:

1. Execute the steps described in the **Getting Started with Linux Kernel and Driver Debugging** section to perform Linux kernel debugging.
2. Select **Run -> Debug Configurations** and select the **Debug Linux_DevKit** configuration created at the previous step.

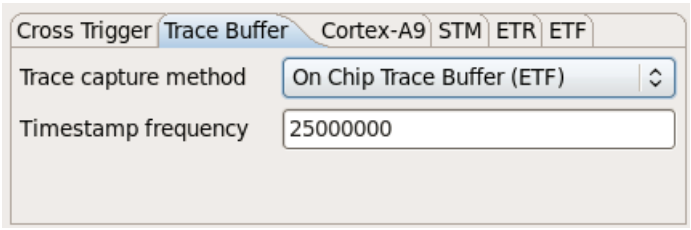
3. Select the **Connection** tab.
4. Click the **Edit DTSL Options** button.

Figure 5-40: Edit DTSL Options Button


Connection
Files
Debugger
OS Awareness
Arguments
Environment
Select the manufacturer, board, project type and debug operation to use. Currently selected:
Altera / Cyclone V SoC / Linux Kernel and/or Device Driver Debug / Debug Cortex-A9x2 SMP via Altera U
Filter platforms
Linux Kernel and/or Device Driver Debug
Debug Cortex-A9_0 via Altera USB-Blaster
Debug Cortex-A9_0 via DSTREAM/RVI
Debug Cortex-A9_1 via Altera USB-Blaster
Debug Cortex-A9_1 via DSTREAM/RVI
Debug Cortex-A9x2 SMP via Altera USB-Blaster
DTSL Options
Edit...
Configure trace or other target options. Using "default" configuration options
DS-5 Debugger will connect to an Altera USB-Blaster to debug a SMP Linux kernel

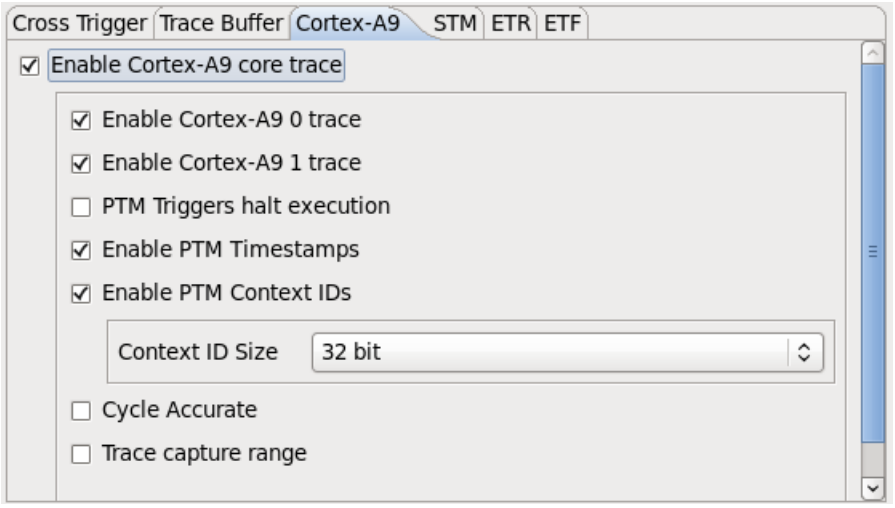
5. In the **DTSL** window**DTSL** dialog box, click **Trace Buffer** tab and select **On Chip Trace Buffer (ETF)** for the **Trace capture method**.

Figure 5-41: Trace Into ETF



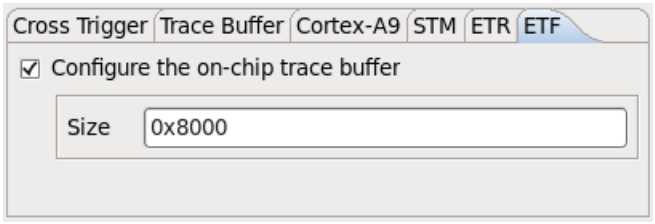
6. In the **DTSL** dialog box, click the **Cortex-A9** tab, and enable tracing for both cores.

Figure 5-42: Enable PTM Tracing



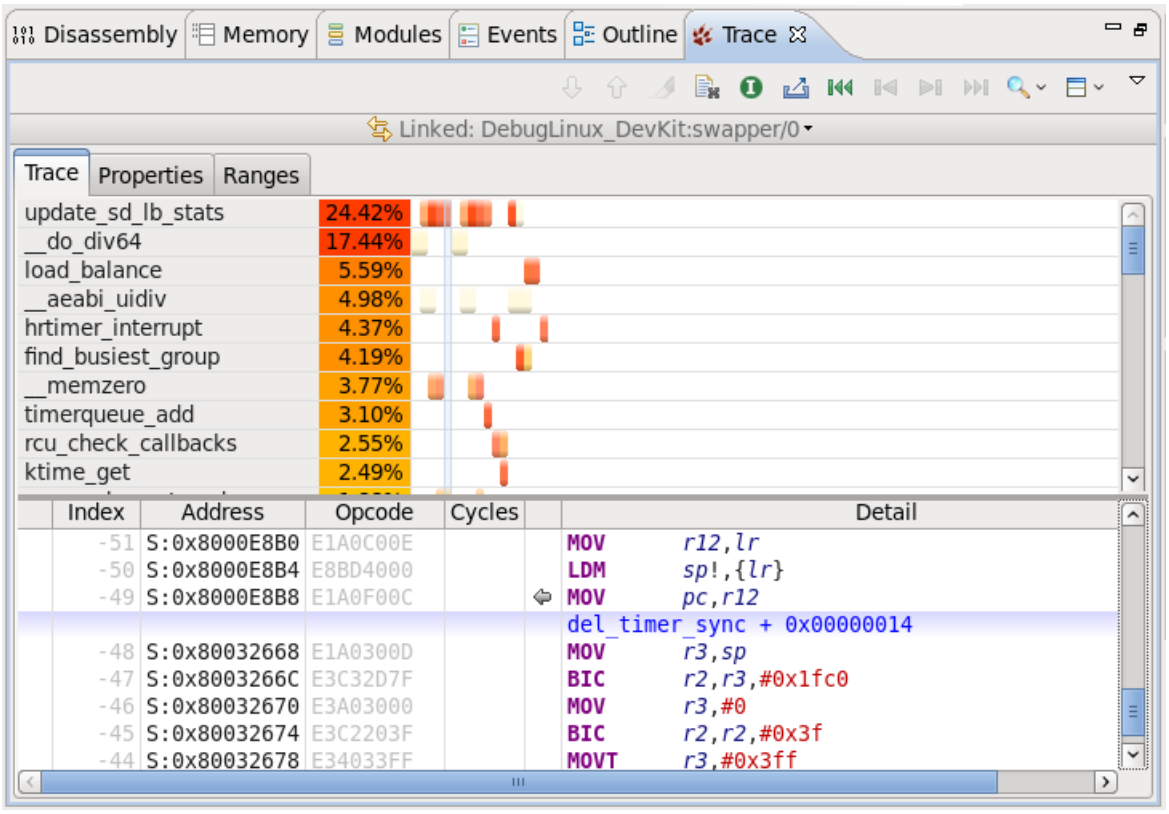
7. In the **ETF** tab, select the ETF to be enabled; and a buffer size of 0x8000, to match the ETF size on the Cyclone V SoC, which is 32 KB.

Figure 5-43: Configure ETF



8. Click **OK** to exit the **DSTL Configuration Editor**.
9. Start a debugging session by starting the **Debug Linux_DevKit** debug configuration. The debugger stops the Linux kernel and configure tracing.
10. Let the kernel run by clicking the **Continue** green button or pressing **F8**.
11. After executing some commands from the Linux serial terminal, click **Interrupt** button or press **F9**. The Debugger shows the captured trace information.

Figure 5-44: DS-5 Debug



The tracing window shows:

- core instructions as they were executed
- percentage occupied by each function
- histogram with function allocation

Related Information

- **ARM DS-5 Altera Edition** on page 6-1
  For more information, refer to the *ARM DS-5 Altera Edition* section.
- **Cyclone V Coresight Debug and Trace**
  For more information about Tracing, refer to the Coresight Debug and Trace section in volume 3 of the *Cycone V Device Handbook*.
- **ARM DS-5 Documentation**
  The ARM DS-5 Altera Edition reference material can be accessed online on the documentation page of the ARM website (www.arm.com); and from Eclipse by navigating to **Help > Help Contents > ARM DS-5 Documentation**.
- **Rocket Boards**
  For more information about Linux, refer to the Rocketboards website.

Getting Started with Cross Triggering

The Altera SoC offers powerful cross-triggering capability between the HPS and the FPGA fabric. The HPS can trigger the FPGA and also the FPGA can trigger the HPS.

ARM has updated the DS-5 tool specifically for Altera to enable this SoC capability to be easily used.

This section presents an example of how cross-triggering can be used.

The Golden Hardware Reference Design (GHRD) contains the necessary instrumentation to be able to use Quartus Signal Tap II tool to demonstrate cross-triggering.

The Quartus Signal Tap II utility is an optional component of the SoC EDS installation, and is selected by default.

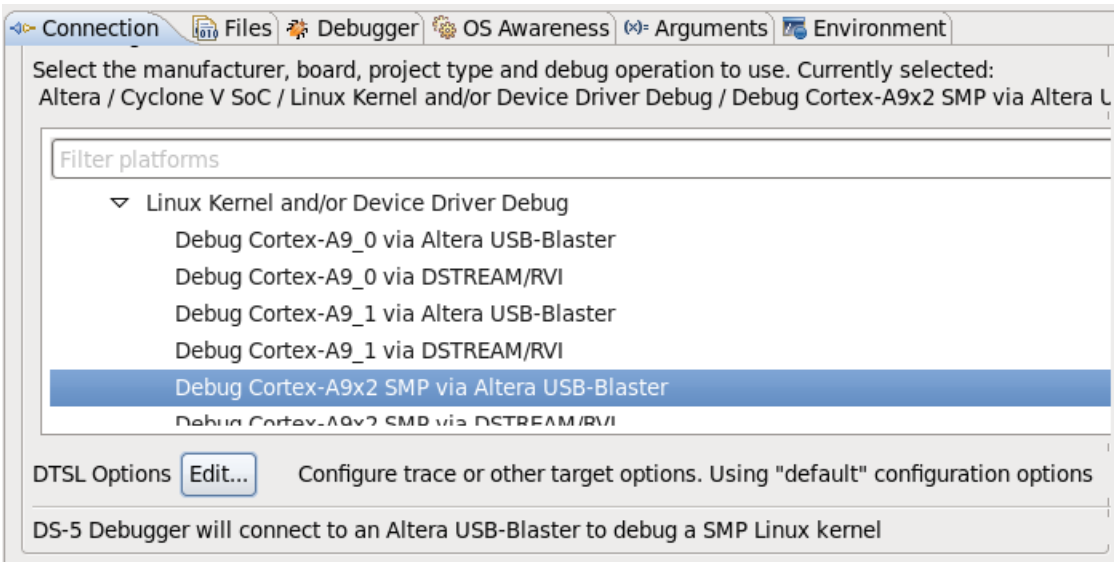
Related Information

- **ARM DS-5 Documentation**
  The ARM DS-5 Altera Edition reference material can be accessed online on the documentation page of the ARM website (www.arm.com); and from Eclipse by navigating to **Help > Help Contents > ARM DS-5 Documentation**.
- **Cyclone V Coresight Debug and Trace**
  For more information about Cross Triggering, refer to the Coresight Debug and Trace section in volume 3 of the *Cycone V Device Handbook*.
- **ARM DS-5 Documentation**
  The ARM DS-5 Altera Edition reference material can be accessed online on the documentation page of the ARM website (www.arm.com); and from Eclipse by navigating to **Help > Help Contents > ARM DS-5 Documentation**.

Enabling Cross-triggering on HPS

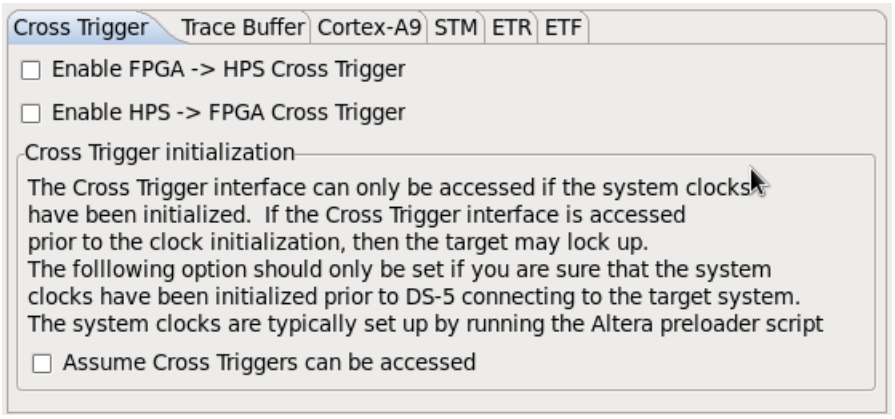
1. Cross-triggering can be enabled in the **DTSL** dialog box, which can be accessed from the **Connection** tab of the **Debug Configuration**.

Figure 5-45:


Connection
Files
Debugger
OS Awareness
Arguments
Environment
Select the manufacturer, board, project type and debug operation to use. Currently selected:
Altera / Cyclone V SoC / Linux Kernel and/or Device Driver Debug / Debug Cortex-A9x2 SMP via Altera U
Filter platforms
Linux Kernel and/or Device Driver Debug
Debug Cortex-A9_0 via Altera USB-Blaster
Debug Cortex-A9_0 via DSTREAM/RVI
Debug Cortex-A9_1 via Altera USB-Blaster
Debug Cortex-A9_1 via DSTREAM/RVI
Debug Cortex-A9x2 SMP via Altera USB-Blaster
DTSL Options
Edit...
Configure trace or other target options. Using "default" configuration options
DS-5 Debugger will connect to an Altera USB-Blaster to debug a SMP Linux kernel

2. The Cross-Trigger tab of the **DTSL Configuration Editor** allows Cross-trigger configuration.

Figure 5-46: HPS Cross-Trigger Configuration


Cross Trigger
Trace Buffer
Cortex-A9
STM
ETR
ETF
Enable FPGA -> HPS Cross Trigger
Enable HPS -> FPGA Cross Trigger
Cross Trigger initialization
The Cross Trigger interface can only be accessed if the system clocks
have been initialized. If the Cross Trigger interface is accessed
prior to the clock initialization, then the target may lock up.
The folllowing option should only be set if you are sure that the system
clocks have been initialized prior to DS-5 connecting to the target system.
The system clocks are typically set up by running the Altera preloader script
Assume Cross Triggers can be accessed

In order to allow FPGA cross triggers to trigger HPS, you need to:

- Check the **Enable FPGA > HPS Cross Triggering** check box
- Check the **Assume Cross Triggers can be accessed** check box

In order to allow HPS cross triggers to trigger FPGA, you need to:

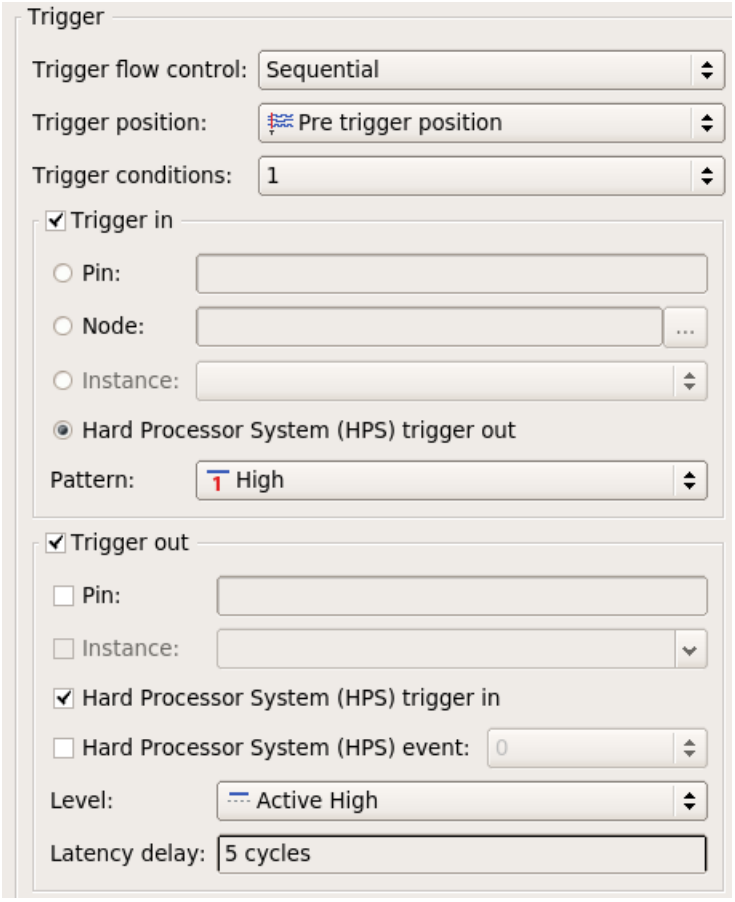
- Check the **Enable HPS > FPGA Cross Triggering** check box
- Check the **Assume Cross Triggers can be accessed** check box

Note: In order to enable bi-directional triggering, you can check all three checkboxes.

Enabling Cross-triggering on FPGA

For this getting started scenario, we are using Quartus SignalTap II utility to control FPGA cross-triggering.

Figure 5-47: SignalTap Cross Trigger Options



Quartus SignalTap II GUI has the above depicted **Trigger** panel that controls cross-triggering:

- The **Trigger In** panel determines whether HPS can trigger FPGA. **Trigger In** can be enabled and the **Pattern** can be selected with **Don't care, Low, High, Rising Edge**, for instance.
- The **Trigger out** panel determines whether FPGA can trigger HPS. **Trigger out** can be enabled and the **Level** can be selected: **Active High** and **Active Low**.

Note: Changing some of the settings requires recompiling the FPGA design. For this getting started scenario, you will change only options that do not require recompilation.

The SignalTap II file that is provided with the Cyclone V GHRD has only the options that do not require compilation enabled to be edited. If you recompile the design, you can enable all options to be edited by selecting the **Lock Mode** to be **Allow all changes**.

Figure 5-48: SignalTap Lock Mode

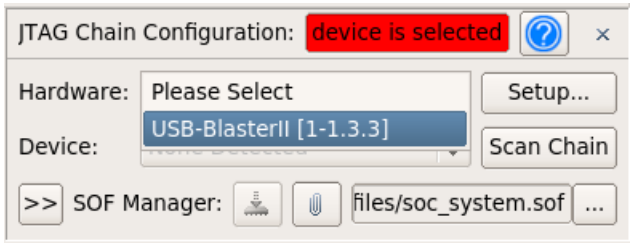
Cross-triggering Prerequisites

This section presents the preparation steps that are required in order to perform the cross triggering scenarios. We boot the HPS, start Signal Tap II and program the FPGA.

Note: Any debugging scenario on HPS can be running, as long as it uses a JTAG connection. It does not necessarily have to be Linux. It could be a bare metal program, for example.

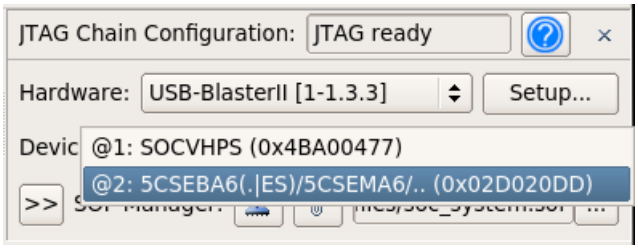
1. Boot the board using the Linux SD card as shown in the *Getting Started with Running Linux* section.
2. Connect USB cable from the **USB Blaster II** connection to the host PC.
3. Open the **Quartus SignalTap II** program by running the command <*SoC EDS installation directory*>**/../qprogrammer/bin/quartus_stpw**. This assumes you have accepted the default settings when installing SoC EDS.
4. In **Signal Tap II**, select File > Open, browse to <*SoC EDS Installation directory*>**/examples/hardware/cv_soc_devkit_ghrd/cti_tapping.stp** and click **Open**
5. In **Signal Tap II**, on the **JTAG Chain Configuration** > **Hardware**, select the USB Blaster II Instance

Figure 5-49: Select USB Blaster II Instance



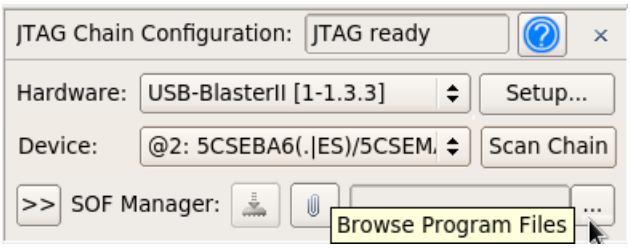
6. In **SignalTap II**, on the **JTAG Chain Configuration** > **Device**, select the FPGA device.

Figure 5-50: Select FPGA device



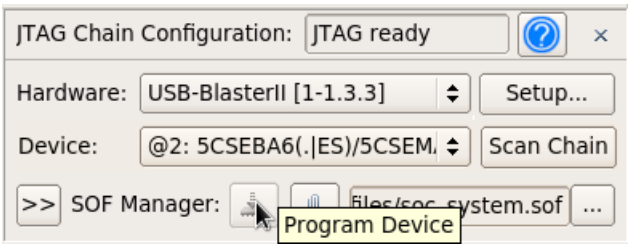
7. In **SignalTap II**, under **SOF Manager**, click the Browse **"..."** button, browse to the file *<SoC EDS Installation directory>***/examples/hardware/cv_soc_devkit_ghrd/output_files/soc_system.sof** and click **Open**.

Figure 5-51: Browse Program Files



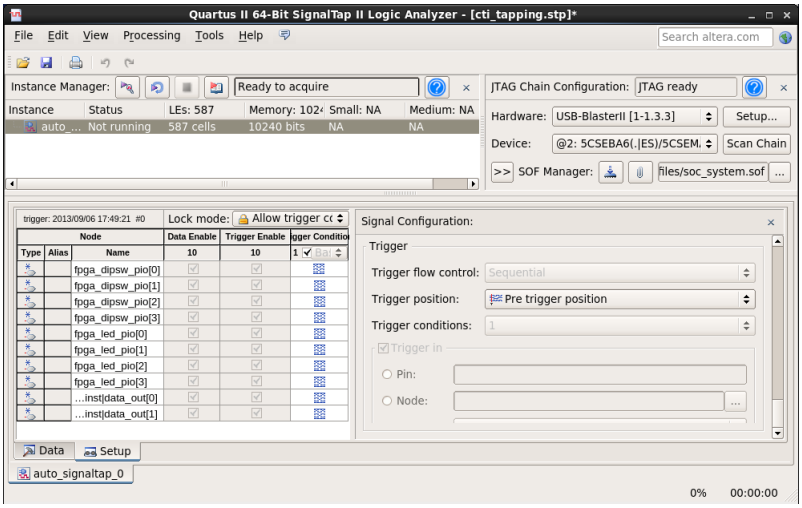
8. In **Signal Tap II**, under **SOF Manager**, click the **Program** button to program the FPGA

Figure 5-52: Program FPGA



9. After the FPGA is programmed, the **SignalTap II** will be ready to acquire:

Figure 5-53: Signal Tap Ready to Acquire



Related Information

Getting Started with Running Linux on page 5-2
For more information, refer to the *Getting Started with Running Linux* section in this document.

FPGA Triggering HPS Example

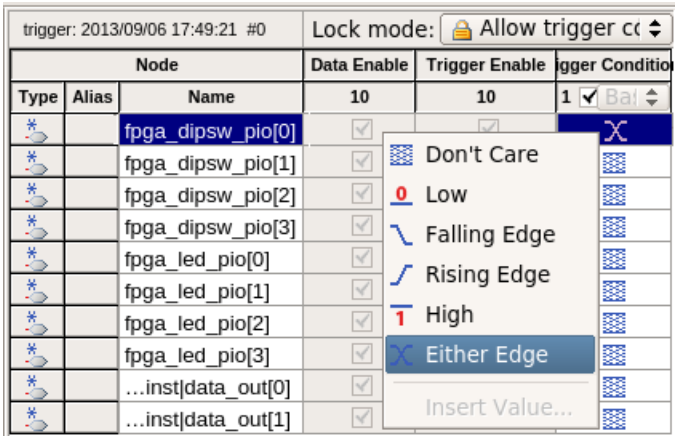
This section presents an example on how FPGA can trigger HPS to stop the execution.

This can be useful if you want to see what the HPS is doing at the moment the trigger comes from FPGA.

The required steps are to reproduce this scenario are:

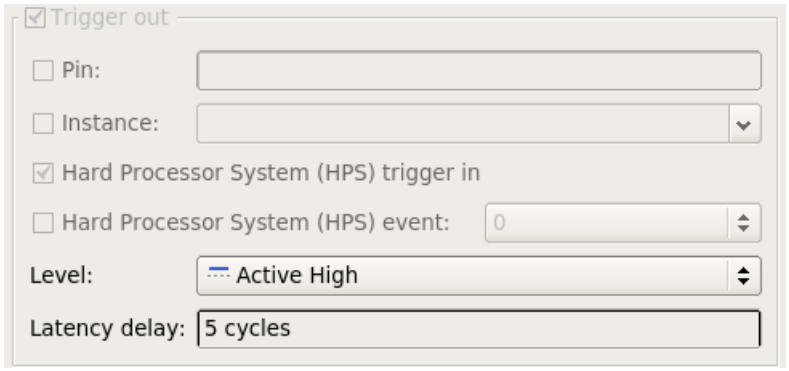
1. Perform the steps from the *Cross-triggering Prerequisites* section.
2. Open the Debugger configuration and edit DTSL options to enable FPGA cross-triggering HPS as shown in the *Enabling Cross-triggering on HPS* section.
   a. Check the **Enable FPGA -> HPS Cross Triggering** check box.
   b. Un-check the **Enable HPS-> FPGA Cross Triggering** check box if checked.
   c. Check the **Assume Cross Triggers can be accessed** check box.
3. Start the debug session by clicking **Debug** in the **Debug Configuration** dialog box. The debugger will stop the Linux kernel and display the current HPS state.
4. In **Signal Tap II**, configure a trigger on the **fpga_dispw_pio[0]** signal to trigger at any edge, by right-clicking the corresponding cell in the **Trigger Condition** column.

Figure 5-54: Trigger on Dip Switch



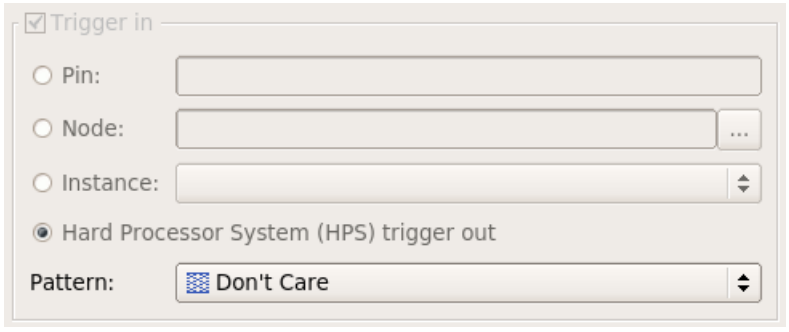
5. In **Signal Tap II**, configure the **Trigger out** to be sent to HPS, so that the **SignalTap II** sends the trigger to HPS whenever it performs an acquisition.

Figure 5-55: Enable Trigger Out to HPS



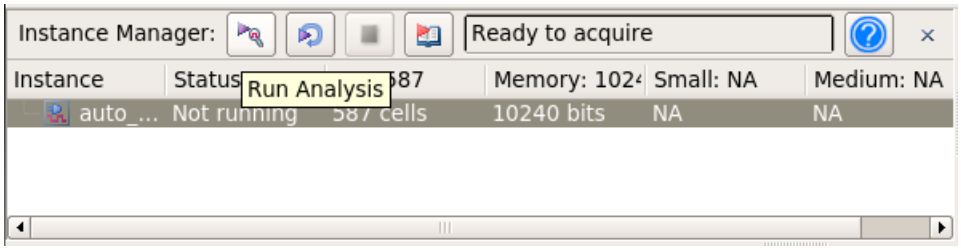
6. In **Signal Tap II**, configure the **Trigger in** to be disabled by setting its pattern to **Don't Care**. In this scenario we do not want the HPS to trigger FPGA.

Figure 5-56: Disable SignalTap Trigger in



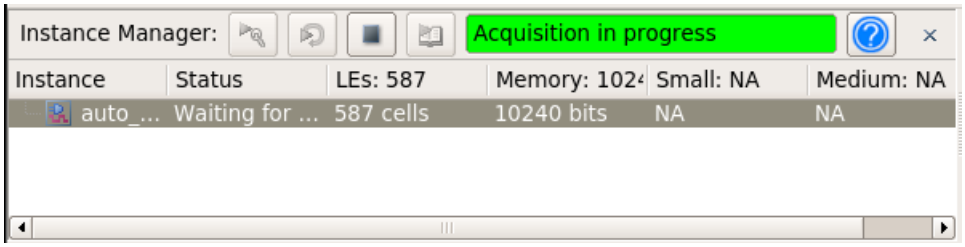
7. In **Eclipse** debugger, let the Linux kernel continue running, by pressing the green **Continue** button or pressing **F8**.
8. In **SignalTap II**, press the **Run Analysis** button to arm Signal Tap:

Figure 5-57: Run Analysis



9. SignalTap II will run the analysis and wait for the trigger from the dipswitch:

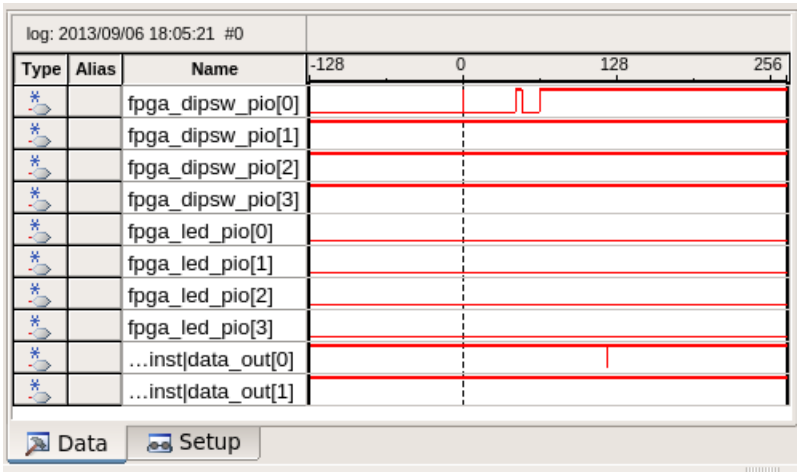
Figure 5-58: Acquisition in Progress



10. Change the state of the FPGA Dipswitch 0 (SW1.5 on the board). This will trigger a **Signal Tap II** acquisition and stop Signal Tap II. This will be indicated by the status changing back to **Ready to acquire**.

Note: On the Data tab, you will be able to see the change on the dipswitch signal.

Figure 5-59: Dipswitch Toggled



11. Go back to the **Eclipse** debugger; you will notice the execution has stopped. When **SignalTap II** is triggered because of the dipswitch, it sends the trigger to HPS, which in turn stops the cores, as instructed.

Related Information

- **Cross-triggering Prerequisites** on page 5-44
  For more information, refer to the *Cross-triggering Prerequisites* section in this document.
- **Enabling Cross-triggering on HPS** on page 5-41
  For more information, refer to the *Enabling Cross-triggering on HPS* section in this document.

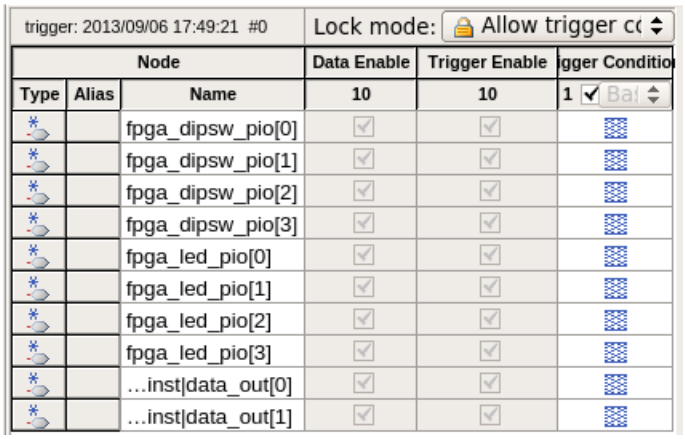
HPS Triggering FPGA Example

This section presents an example on how stopping HPS execution in the debugger can trigger FPGA to perform a **SignalTap II** acquisition. This can be useful, for example, if we want to see the state of some FPGA signals at the time the HPS is stopped in the debugger.

The required steps are to reproduce this scenario:

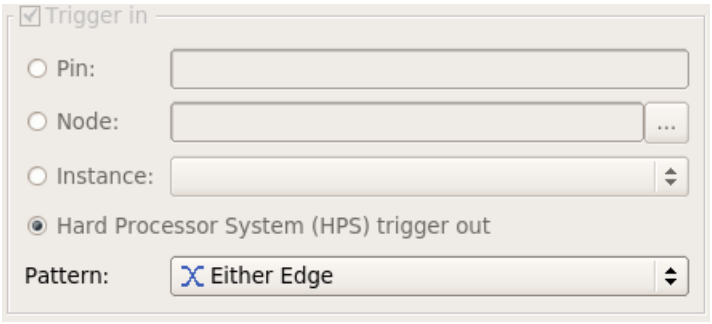
1. Perform the steps from the *Cross-triggering Prerequisites* section.
2. Open the Debugger configuration and edit DTSL options to enable FPGA cross-triggering HPS as shown in the *Enabling Cross-triggering on HPS* section.
   a. Un-check the **Enable FPGA -> HPS Cross Triggering** check box.
   b. Check the **Enable HPS-> FPGA Cross Triggering** check box if checked.
   c. Check the **Assume Cross Triggers can be accessed** check box.
3. Start the debug session by clicking **Debug** in the **Debug Configuration** dialog box. The debugger will stop the Linux kernel and display the current HPS state.
4. In **Signal Tap II**, make sure all trigger signals are disabled by setting their condition to **Don't care**.

Figure 5-60: Trigger Signals Disabled



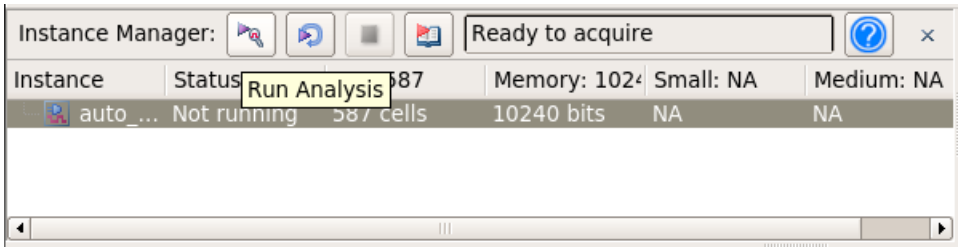
5. In **Signal Tap II**, configure the **Trigger in** to be sensitive to both edges, so that the **SignalTap II** sends the trigger to HPS whenever it performs an acquisition.

Figure 5-61: Configure Trigger in



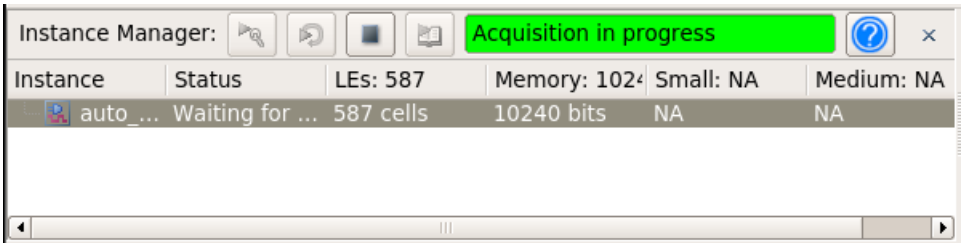
6. In **Eclipse**debugger, let the Linux kernel continue running, by pressing the green **Continue** button or pressing **F8**.
7. In **SignalTap II**, press the **Run Analysis** button to arm Signal Tap:

Figure 5-62: Run Analysis



8. **SignalTap II** will run the analysis and wait for the trigger from HPS:

Figure 5-63: Acquisition in Progress



9. In **Eclipse** debugger, click the **Interrupt** button or press **F9**. This will stop the cores and send the trigger to FPGA.
10. **SignalTap II** will detect the trigger from HPS, perform an acquisition and stop. This will be indicated by the status changing back to **Ready to acquire**.

Related Information

- **ARM DS-5 Altera Edition** on page 6-1
  For more information, refer to the *ARM DS-5 Altera Edition* section.

- **Cyclone V Coresight Debug and Trace**
  For more information about Tracing, refer to the Coresight Debug and Trace section in volume 3 of the *Cycone V Device Handbook*.
- **ARM DS-5 Documentation**
  The ARM DS-5 Altera Edition reference material can be accessed online on the documentation page of the ARM website (www.arm.com); and from Eclipse by navigating to **Help > Help Contents > ARM DS-5 Documentation**.
- **Rocket Boards**
  For more information about Linux, refer to the Rocketboards website.

ARM DS-5 Altera Edition 6

Subscribe Send Feedback

The ARM DS-5 Altera Edition is based on the ARM Development Studio 5 (DS-5) Toolkit and is a device-specific exclusive offering from Altera.

The ARM DS-5 Altera Edition is a powerful Eclipse-based comprehensive Integrated Development Environment. Some of the most important provided features are:

- File editing, supporting syntax highlighting and source code indexing
- Build support, based on makefiles
- Bare-metal debugging
- Linux application debugging
- Linux kernel and driver debugging
- Multicore debugging
- Access to HPS peripheral registers
- Access to FPGA soft IP peripheral registers
- Tracing of program execution through PTM
- Tracing of system events through STM
- Cross-triggering between HPS and FPGA
- Connecting to the target using Altera USB Blaster II

The ARM DS-5 Altera Edition is a complex tool with a vast amount of features and options. The Altera SoC EDS User Guide only describes some of the most common features and options and provides getting started scenarios to allow you to start being productive, quickly.

Related Information

ARM DS-5 Documentation

The ARM DS-5 Altera Edition reference material can be accessed online on the documentation page of the ARM website (www.arm.com); and from Eclipse by navigating to **Help > Help Contents > ARM DS-5 Documentation**.

Starting Eclipse

Eclipse is the IDE used by ARM DS-5 Altera Edition and it can be started from the Embedded Command Shell or from the windows file menu selection.

The advantage of starting Eclipse from the Embedded Command Shell is that all the utilities are added to the search path, and they can be used directly from the makefiles without the full path.

1. At the command line, type **eclipse &** to start Eclipse IDE used by ARM DS-5 Altera Edition. See Embedded Command Shell section for more details about the shell.

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

2. On Windows, Eclipse can be started by selecting **Start > All Programs > ARM DS-5 > Eclipse for DS-5**. On different Linux machines, shortcuts may be created for starting Eclipse in a similar manner.

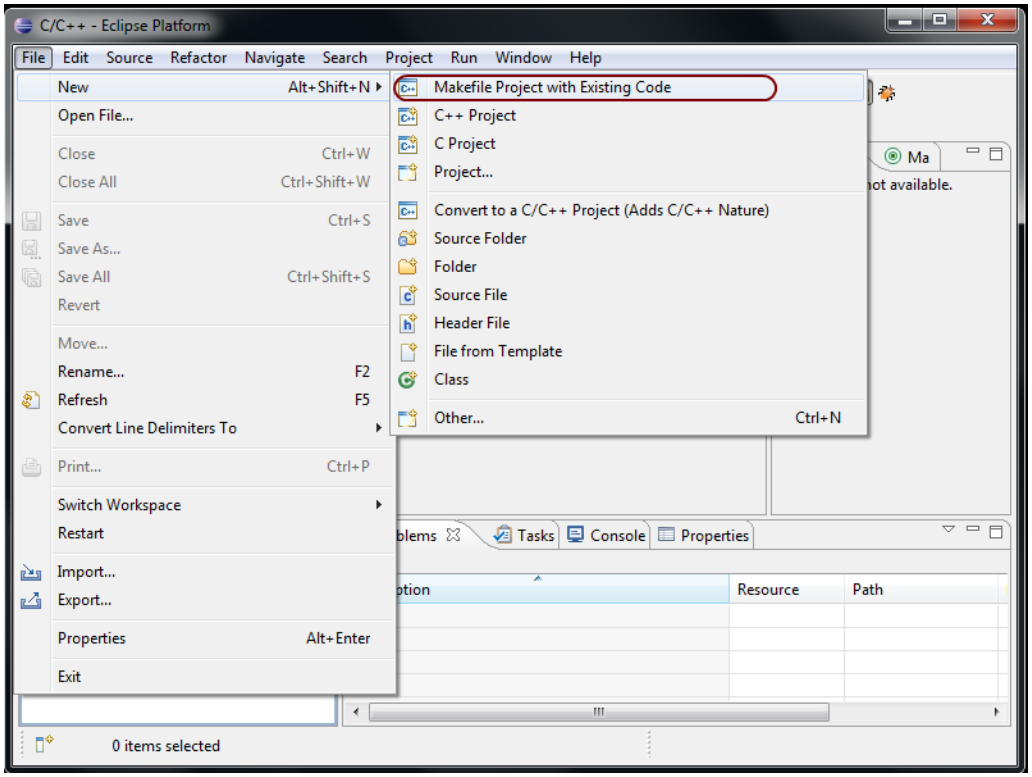
Project Management

ARM DS-5 Altera Edition enables convenient project management using makefiles. The sample projects that are provided with SoC EDS use makefiles to manage the build process.

In order to allow Eclipse to manage a makefile-based project, a project needs to be created:

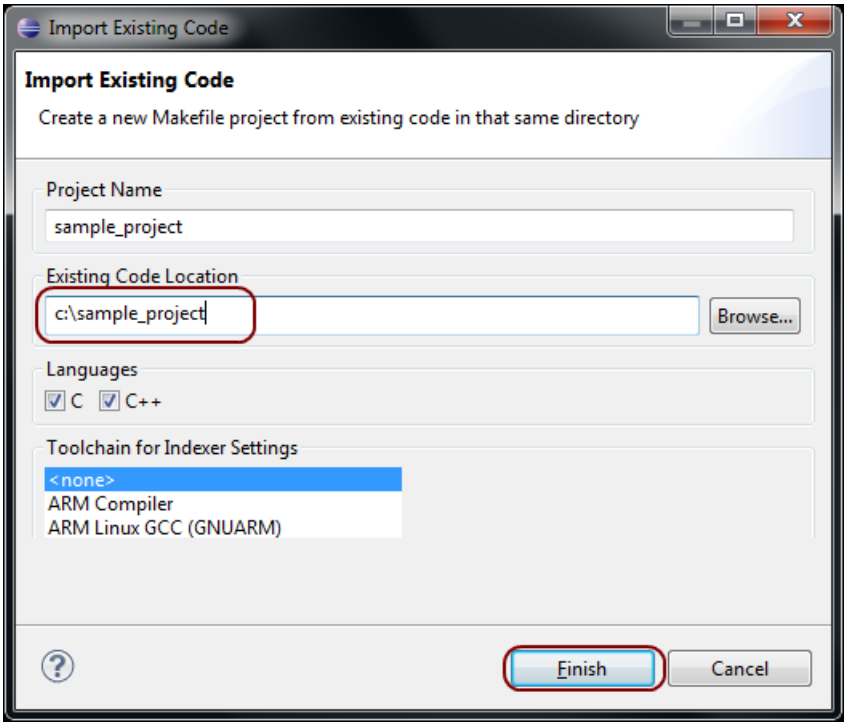
1. Create a folder on the disk. For example, we have created the folder **c:\sample_project**.
2. Create the project by selecting **File > New > Makefile Project with Existing Code**.

Figure 6-1: Creating a Project with Existing Code



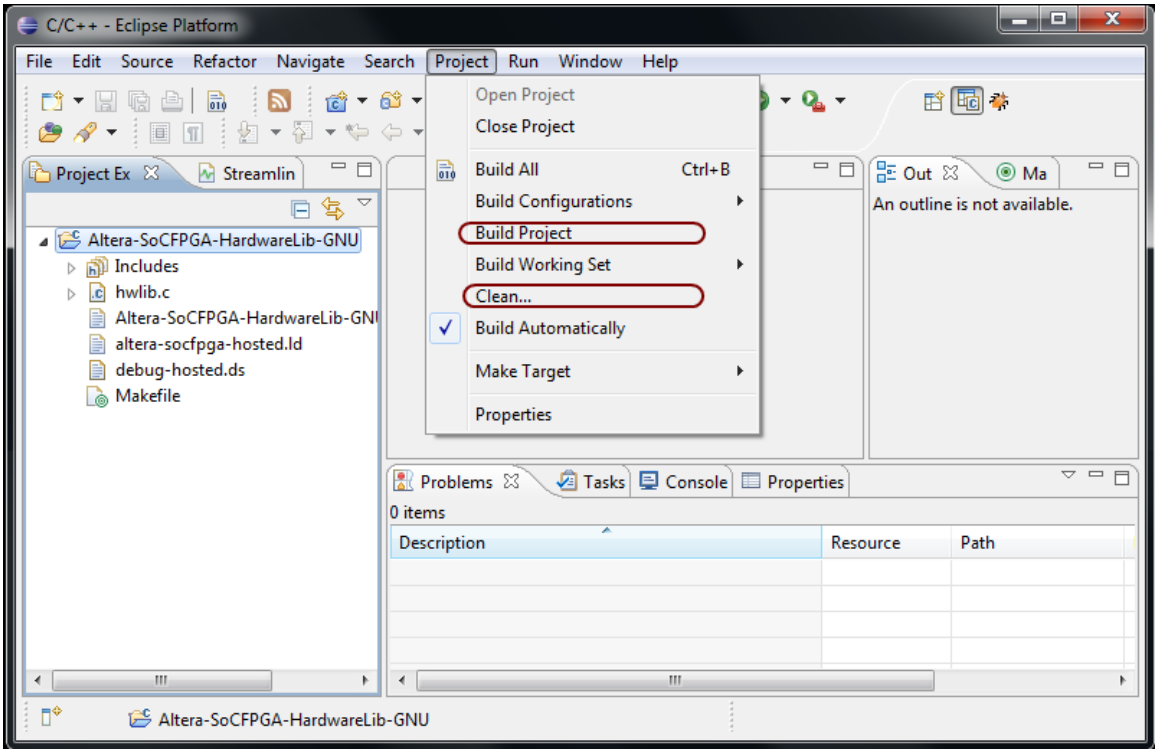
3. Type the folder name in the **Existing Code Location** edit box and then click **Finish**.

Figure 6-2: Import Existing Code window



4. Create a **Makefile** in that folder, and define the rules required for compiling the code. Make sure it has the **all** and the **clean** targets.
5. Eclipse now offers the possibility of invoking the build process from the IDE.

Figure 6-3: Eclipse IDE - build process invoked



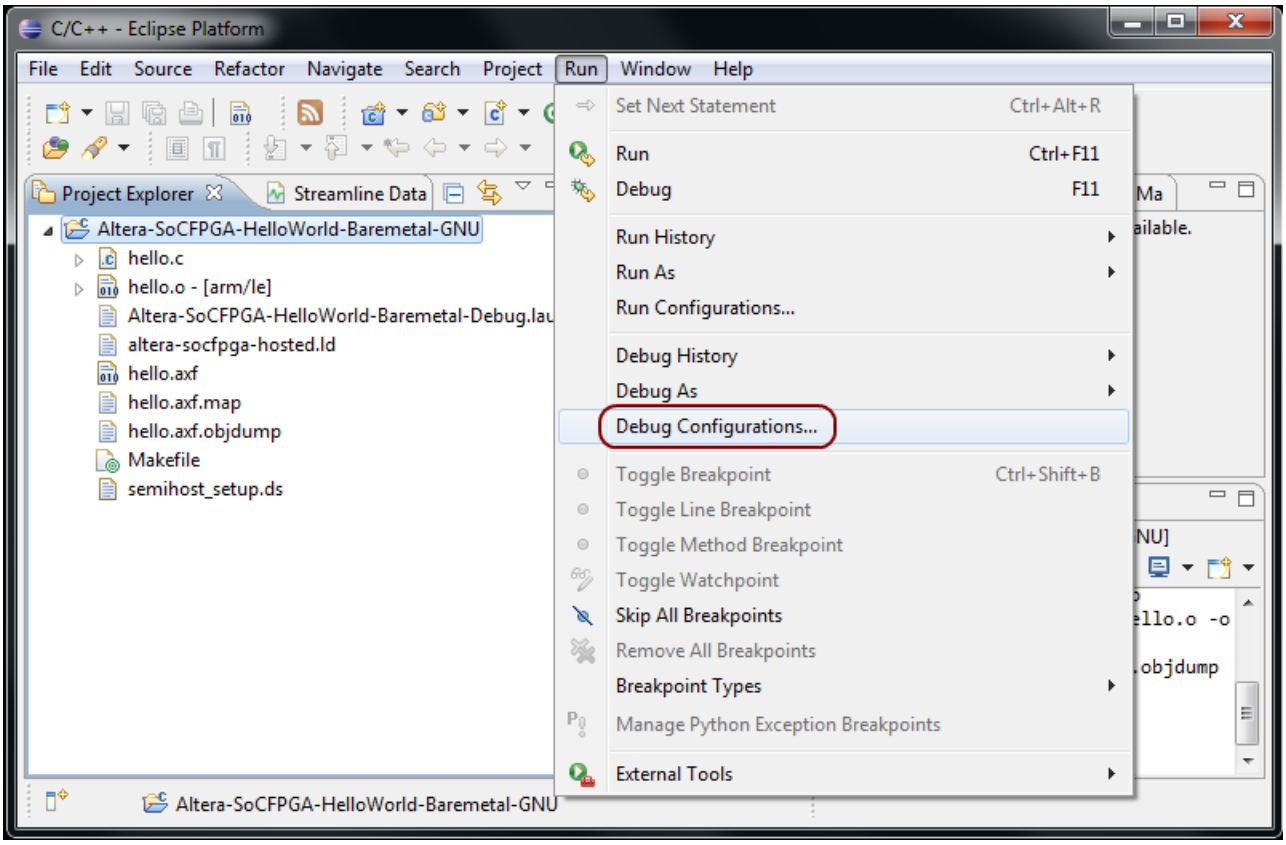
6. If the compilation tools issue errors, Eclipse parses and formats them for you.

Debugging

ARM DS-5 AE offers you a variety of debugging features.

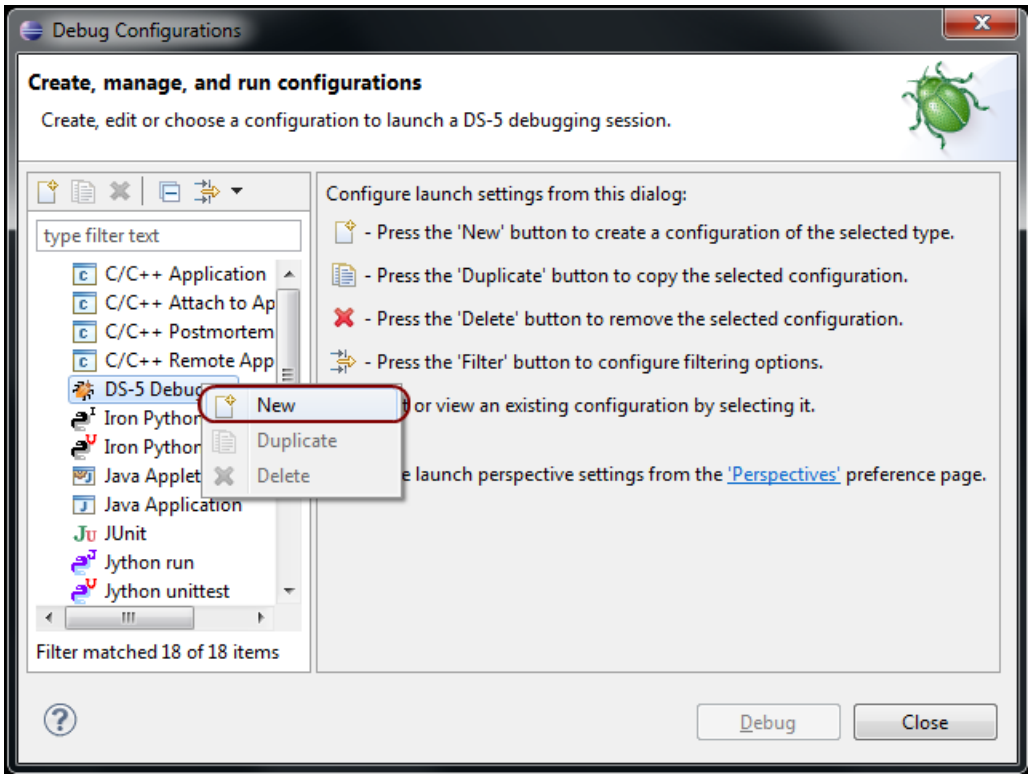
The settings for a debugging session are stored in a **Debug Configuration**. The **Debug Configurations** window is accessible from the **Run > Debug Configurations** menu.

Figure 6-4: Accessing Debug Configurations



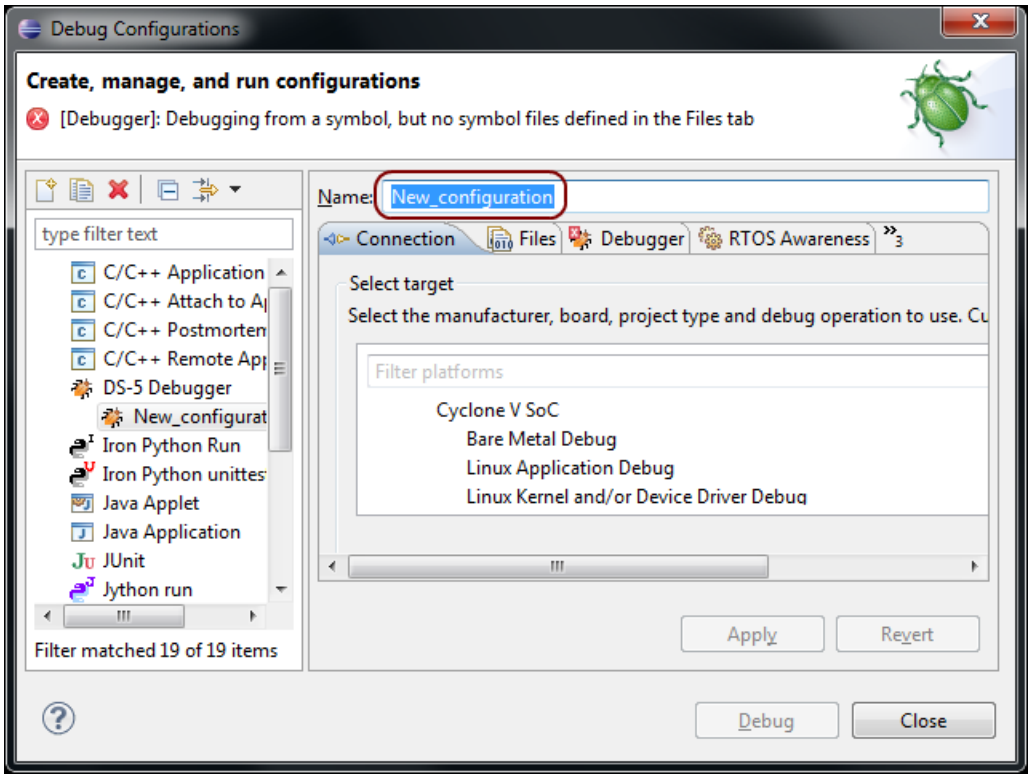
A **Debug Configuration** is created in the **Debug Configurations** window by selecting **DS-5 Debugger** as the type of configuration in the left panel and then right-click and select the **New** menu option.

Figure 6-5: Creating a Debug Configuration


Debug Configurations
Create, manage, and run configurations
Create, edit or choose a configuration to launch a DS-5 debugging session.
type filter text
C/C++ Application
C/C++ Attach to Ap
C/C++ Postmortem
C/C++ Remote App
DS-5 Debug
Iron Python
Iron Python
Java Applet
Java Application
JUnit
Jython run
Jython unittest
Filter matched 18 of 18 items
New
Duplicate
Delete
Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
or view an existing configuration by selecting it.
launch perspective settings from the 'Perspectives' preference page.
Debug
Close

The Eclipse IDE assigns a default name to the configuration, which can then be edited by you.

Figure 6-6: Naming your Configuration



Debug Configuration Options

This section lists the **Debug Configuration** options, which allows you to specify the desired debugging options for a project.

Refer to the DS-5 reference documentation for the complete details.

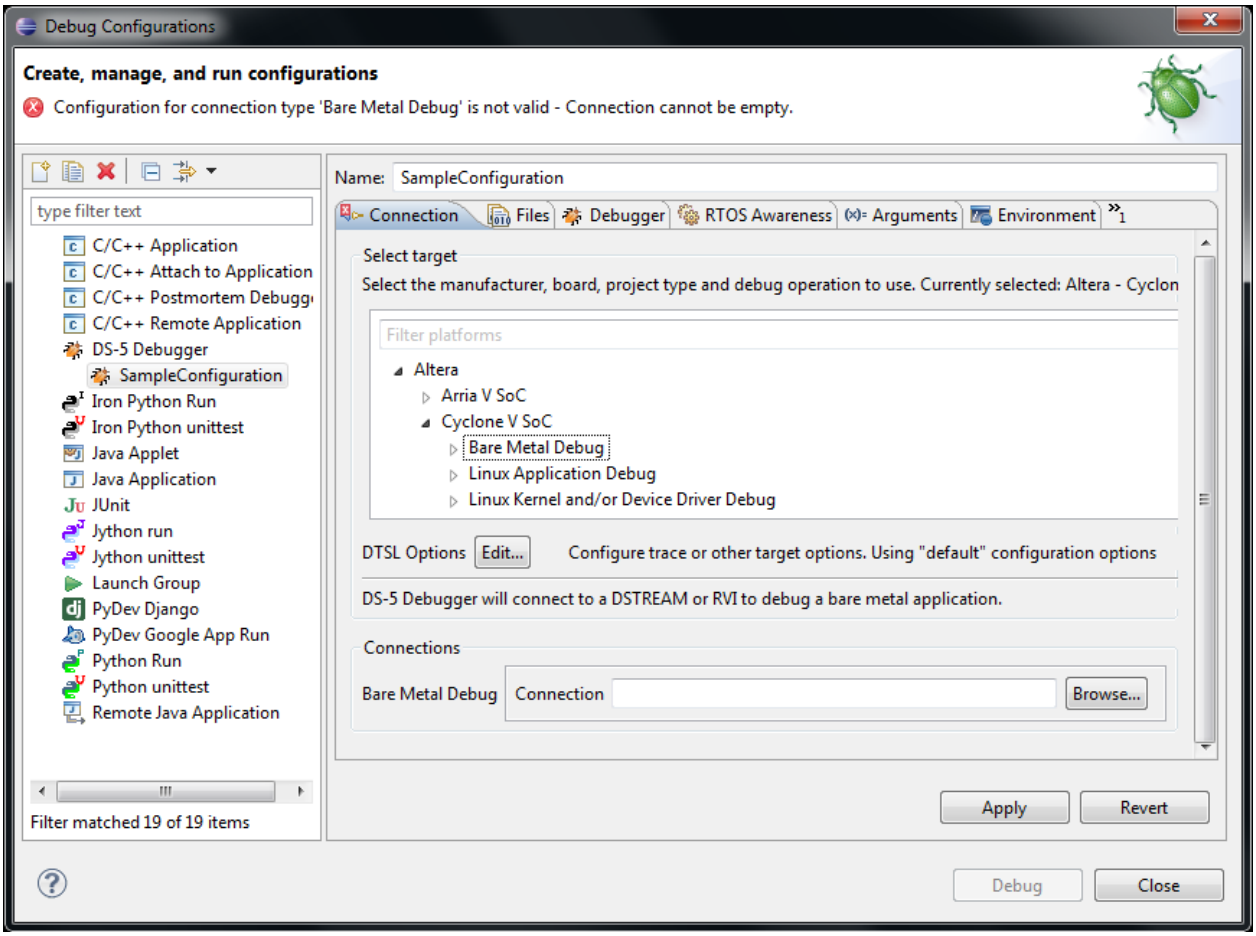
Refer to the **Getting Started Guides** for concrete examples on how to use the debugging features of ARM DS-5 Altera Edition.

Connection Options

The **Connection** tab offers the following configurable options:

- **Target** – specifies which target to use
- **DTSL Options** – specifies the Debugger and (described in detail in another section)
- **Connections** – specifies the way to connect to the target

Figure 6-7: Debug Configurations - Connection



The following **Target Types** are available:

- **Bare Metal Debug** – select this to debug bare metal applications
- **Linux Application Debug** – select this to debug Linux applications
- **Linux Kernel and/or Device Driver Debug** – select this to debug the Linux kernel and Linux device drivers

For the **Bare Metal Debug** and **Linux Kernel and/or Device Driver Debug** target types, the following different scenarios are available:

- **Debug Cortex-A9_0 via Altera USB-Blaster**
- **Debug Cortex-A9_0 via DSTREAM/RVI**
- **Debug Cortex-A9_1 via Altera USB-Blaster**
- **Debug Cortex-A9_1 via DSTREAM/RVI**
- **Debug Cortex-A9x2 SMP via Altera USB-Blaster**
- **Debug Cortex-A9x2 SMP via DSTREAM/RVI**

For **Linux Application Debug** option, the following different scenarios are available:

- **Connect to already running GDB server**
- **Download and debug application**
- **Start gdbserver and debug target resident application**

The **Connections** panel allows the user to specify the connection to be used to communicate to the board.

Depending on the Target type and the selected scenario, the Connections panel will look different:

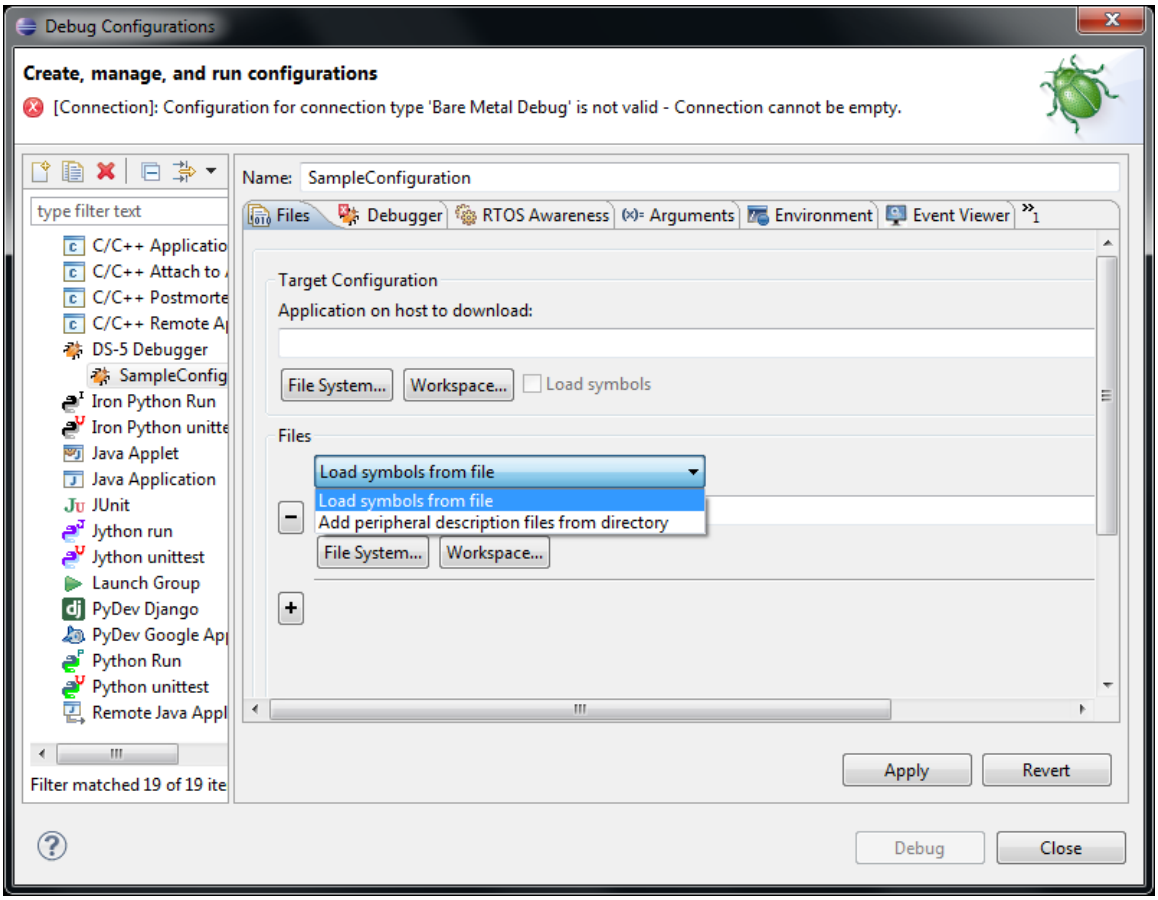
- For **Bare Metal Debug** and **Linux Kernel and/or Device Driver Debug** target types, a **Browse** button appears and it allows the user to browse and select either the **Altera USB-Blaster** or the **DSTREAM** instances.
- For the **Linux Application Debug**, the **Connection** needs to be configured in the **Remote System Explorer** view

File Options

The **Files** tab allows the following settings to be configured:

- **Application on host to download** – the file name of the application to be downloaded to the target. It can be entered directly in the edit box or it can be browsed for in the **Workspace** or on the **File System**.
- **Files** – contains a set of files. A file can be added to the set using the "+" button, and files can be removed from the set using the "–" button. Each file can be one of the following two types:
  - **Load symbols from file** – the debugger uses that file to load symbols from it
  - **Add peripheral description files from directory** – the debugger to load peripheral register descriptions from the **SVD** files stored in that directory. The **SVD** file is a result of the compilation of the hardware project.

Figure 6-8: Debug Configurations - File options

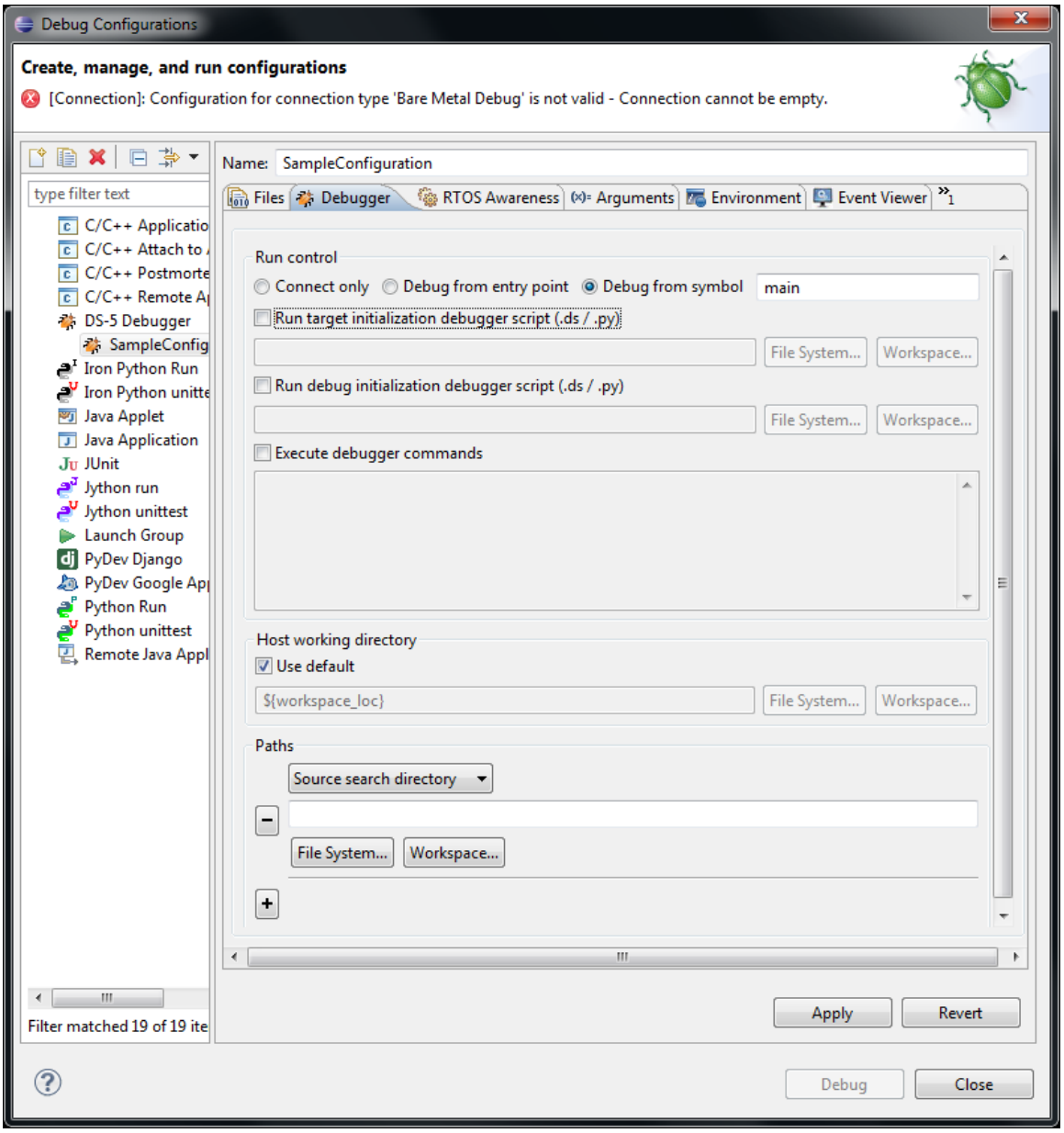


Debugger Options

The **Debugger** tab offers the following configurable options

- **Run Control** Options
  - Option to connect only, debug from entry point or debug from user-defined symbol
  - Option to run user-specified target initialization script
  - Option to run user-specified debug initialization script
  - Option to execute user-defined debugger commands
- **Host working directory** – used by semihosting
- **Paths** – allows you to enter multiple paths for the debugger to search for sources. Paths can be added with “+” button and removed with “-" button.

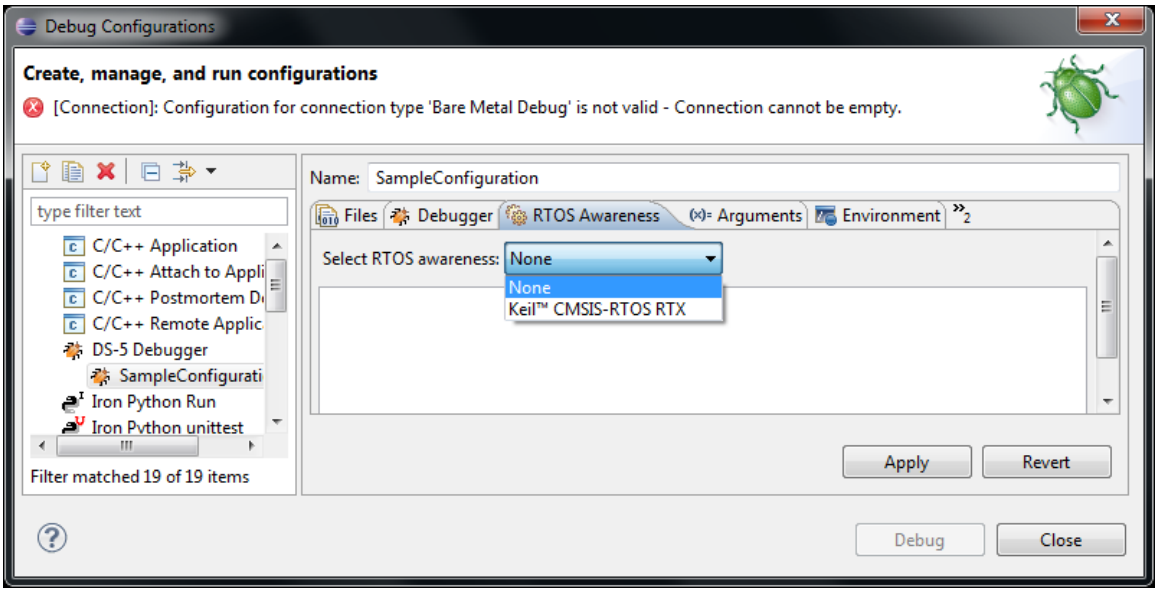
Figure 6-9: Debug Configurations - Debugger



RTOS Awareness

The **RTOS Awareness** tab allows you to enable Keil CMSIS-RTOS RTX awareness for the debugger in case that specific RTOS is used.

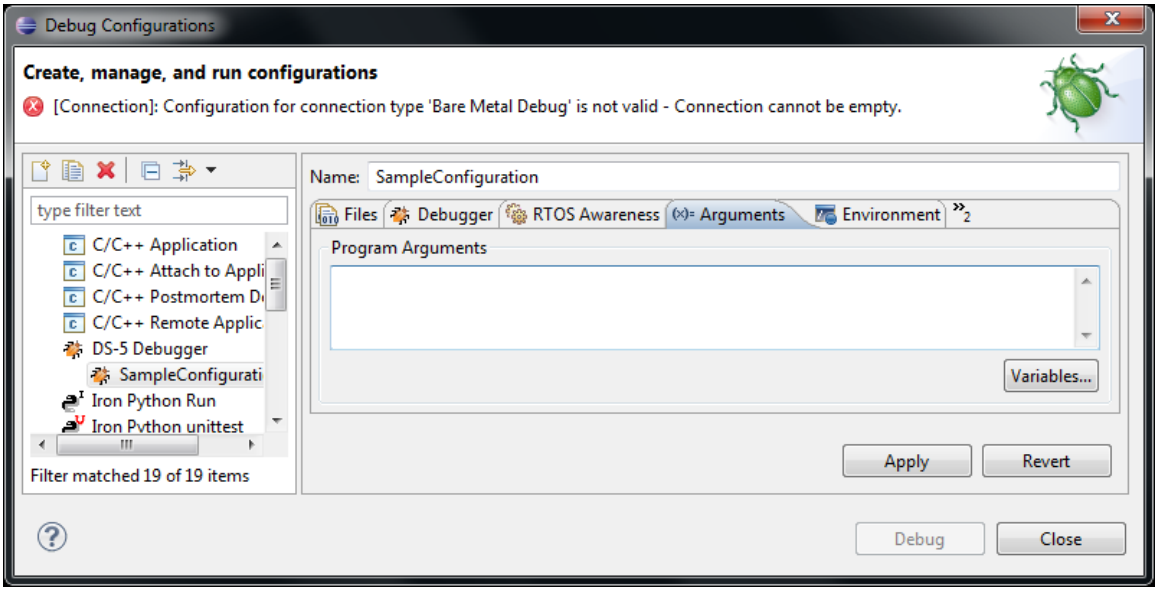
Figure 6-10: Debug Configurations - RTOS Awareness



Arguments

The **Arguments** tab allows you to enter program arguments as text.

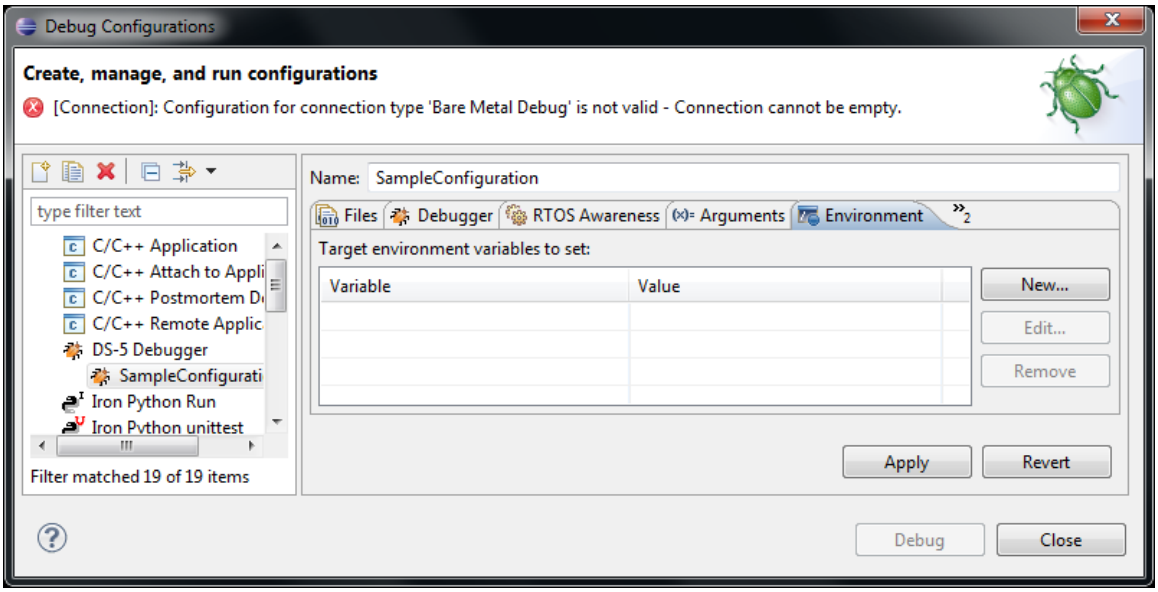
Figure 6-11: Debug Configurations - Arguments



Environment

The **Environment** tab allows you to enter environment variables for the program to be executed.

Figure 6-12: Debug Configurations - Environment

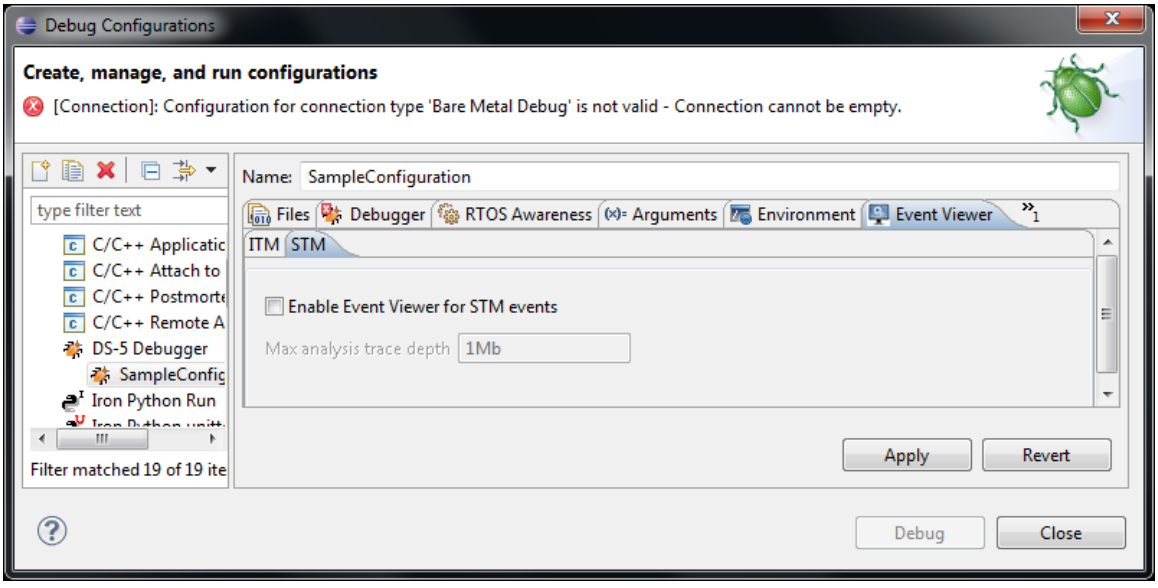


Event Viewer

The **Event Viewer** allows you to enable and configure the event views for the STM (System Trace Macrocell) and ITM (Instrumentation Trace Macrocell).

The SoC FPGA does not have an ITM, so only STM can be enabled. When STM Viewer is enabled, the size of the trace depth is also configured.

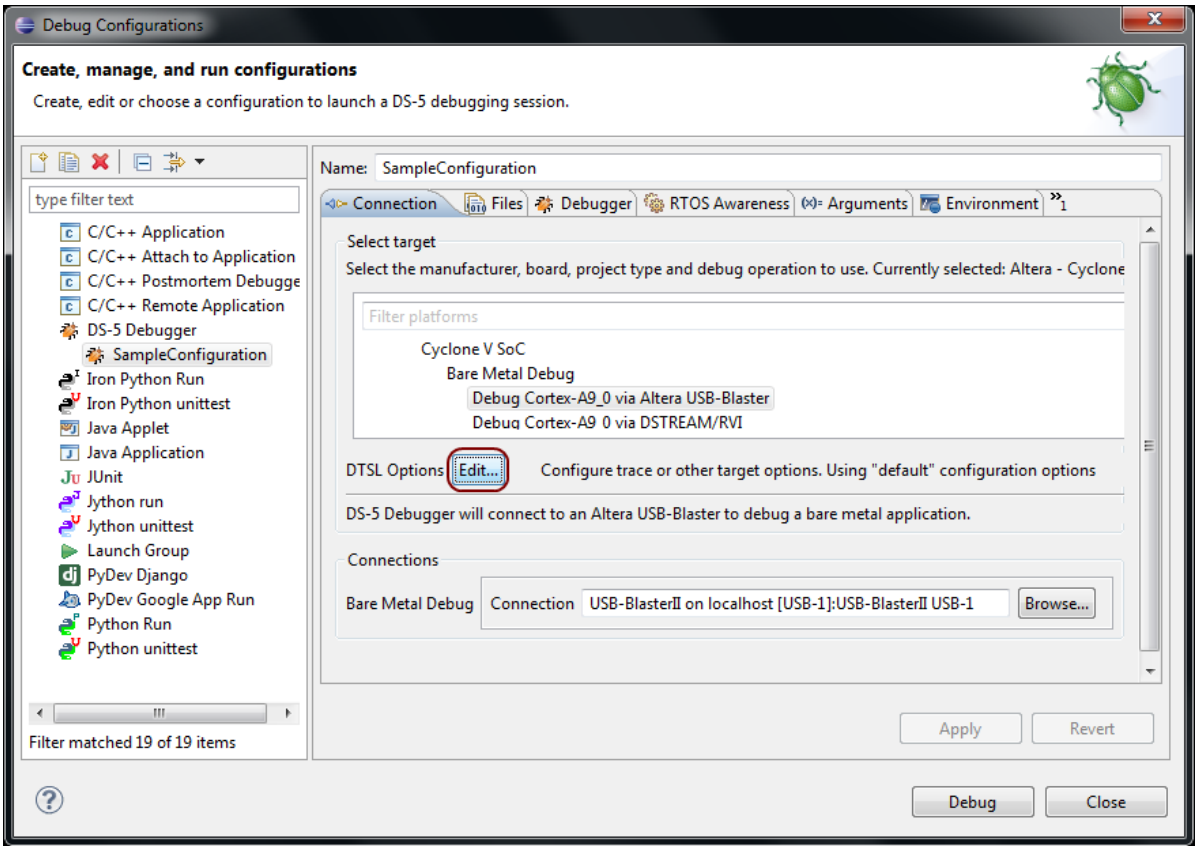
Figure 6-13: Debug Configurations - Event Viewer



DTSL Options

The tracing options are configured from the **DTSL Configuration** window accessible from the **Debug Configuration** window by pressing the **Edit** button. **DTSL** represents **Debug and Trace Services Layer.**

Figure 6-14: Debug Configurations - DTSL Options - Edit



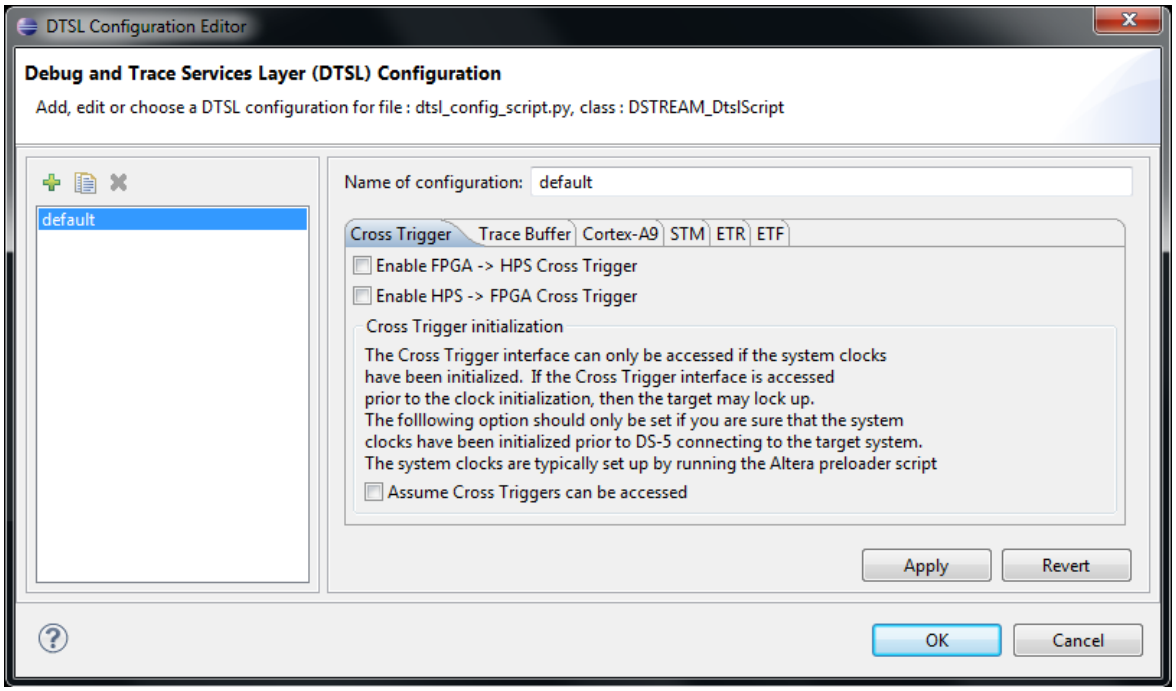
Cross Trigger Settings

The **Cross Trigger** tab allows the configuration of the cross triggering option of the SoC FPGA.

The following options are available:

- **Enable FPGA > HPS Cross Trigger** – for enabling triggers coming from FPGA to HPS
- **Enable HPS > FPGA Cross Trigger** – for enabling triggers coming from HPS to FPGA
- **Assume Cross Triggers can be accessed** – the user needs to select this option as a confirmation that the Preloader was already loaded, so the DS-5 can access the cross triggering interface.

Figure 6-15: DTSL Configuration Editor - Cross Trigger


DTSL Configuration Editor
Debug and Trace Services Layer (DTSL) Configuration
Add, edit or choose a DTSL configuration for file : dtsl_config_script.py, class : DSTREAM_DtslScript
default
Name of configuration: default
Cross Trigger
Trace Buffer
Cortex-A9
STM
ETR
ETF
Enable FPGA -> HPS Cross Trigger
Enable HPS -> FPGA Cross Trigger
Cross Trigger initialization
The Cross Trigger interface can only be accessed if the system clocks
have been initialized. If the Cross Trigger interface is accessed
prior to the clock initialization, then the target may lock up.
The folllowing option should only be set if you are sure that the system
clocks have been initialized prior to DS-5 connecting to the target system.
The system clocks are typically set up by running the Altera preloader script
Assume Cross Triggers can be accessed
Apply
Revert
OK
Cancel

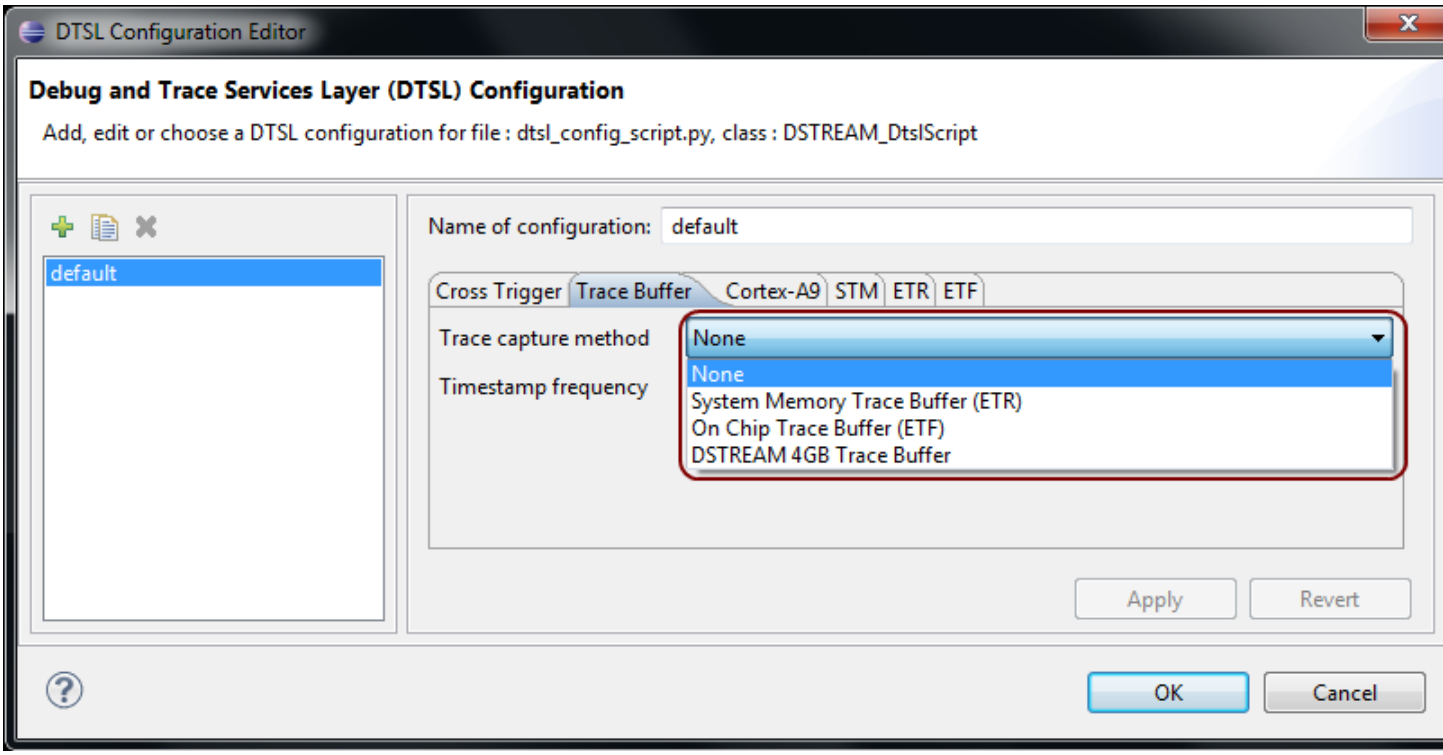
Trace Buffer Settings

The **Trace Buffer** tab allows the selection of the destination of the trace information. As mentioned in the introduction, the destination can be one the following:

- **None** – meaning the tracing is disabled
- **ETR** – using any memory buffer accessible by HPS
- **ETF** – using the 32KB on-chip trace buffer
- **DSTREAM** – using the 4GB buffer located in the DSTREAM

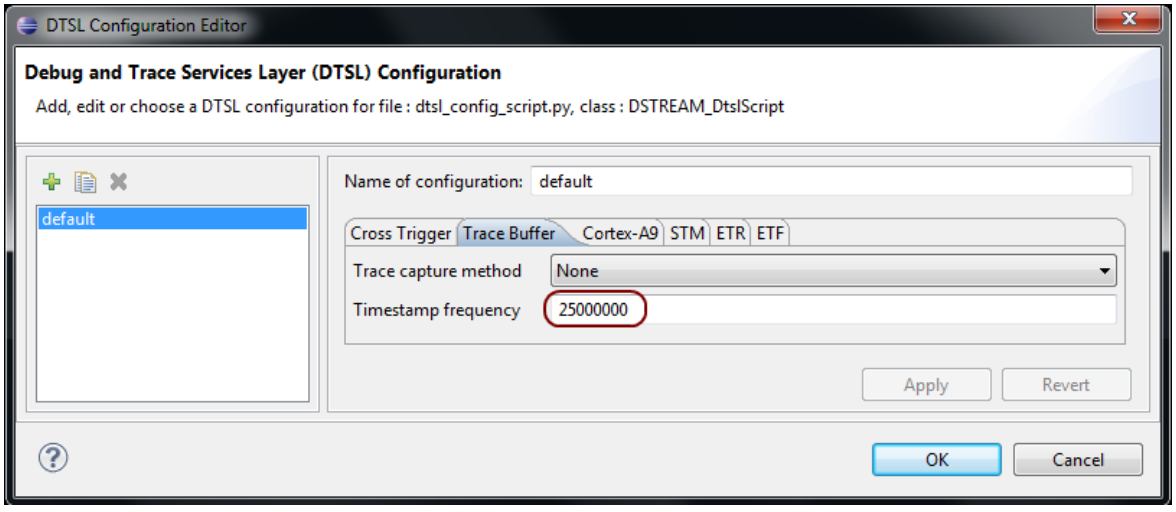
The DSTREAM option is available only if the **Target** connection is selected as **DSTREAM** in the **Debug Configuration**.

Figure 6-16: DTSL Configuration Editor - Trace Buffer



The **Trace Buffer** tab provides the option of selecting the timestamp frequency.

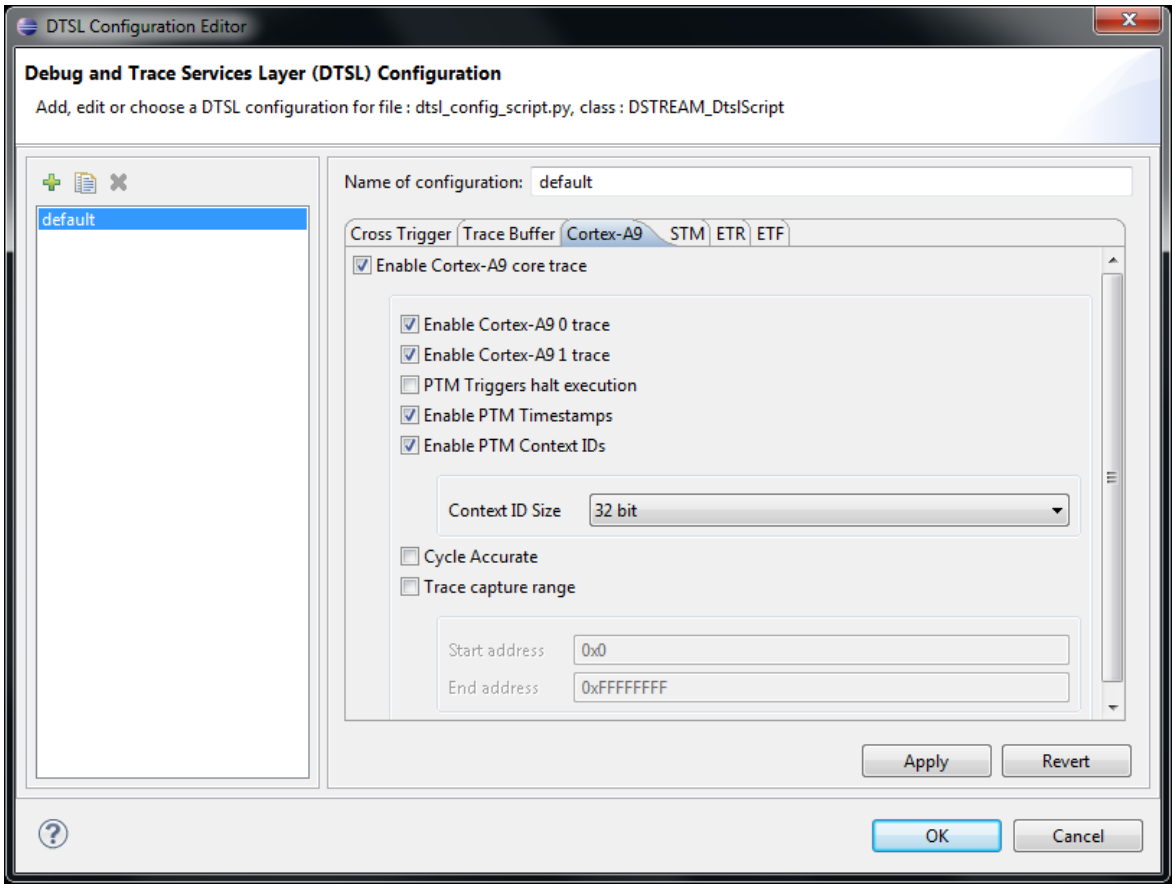
Figure 6-17: DTSL Configuration Editor - Trace Buffer



Cortex-A9 Settings

The **Cortex-A9** tab allows the selection of the desired core tracing options.

Figure 6-18: DTSL Configuration Editor - Cortex-A9



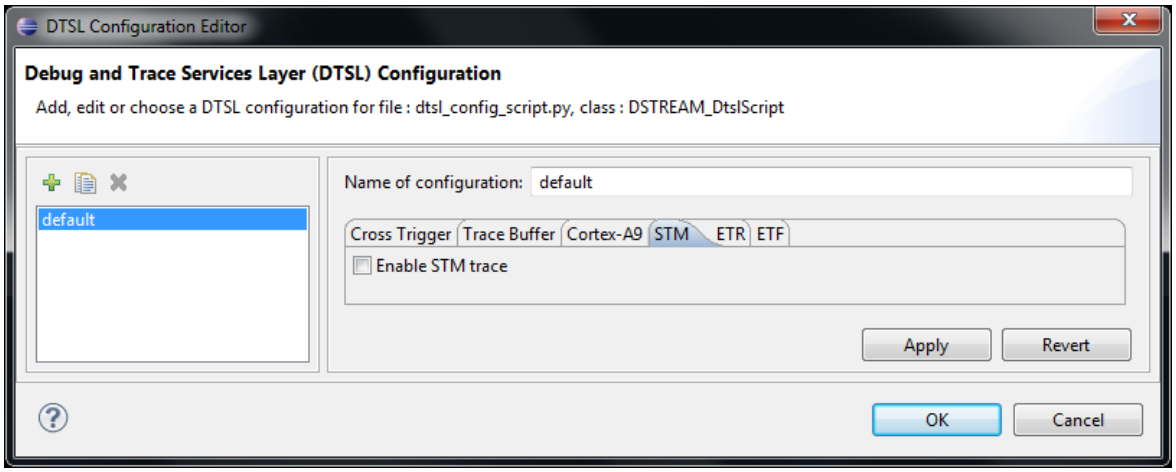
The following **Core Tracing Options** are available:

- **Enable Cortex-A9 0 core trace** – check to enable tracing for core #0
- **Enable Cortex-A9 1 core trace** – check to enable tracing for core #1
- **PTM Triggers halt execution** – check to cause the execution to halt when tracing
- **Enable PTM Timestamps –** check to enable time stamping
- **Enable PMT Context IDs –** check to enable the context IDs to be traced
- **Context ID Size** – select 8, 16 or 32bit context IDs. Used only if Context IDs are enabled
- **Cycle Accurate** – check to create cycle accurate tracing
- **Trace capture range** – check to enable tracing only a certain address interval
- **Start Address, End Address** – define the tracing address interval (Used only if the Trace Capture Range is enabled)

STM Settings

The **STM** tab allows you to configure the STM (System Trace Macrocell).

Figure 6-19: DTSL Configuration Editor - STM



Only one option is available:

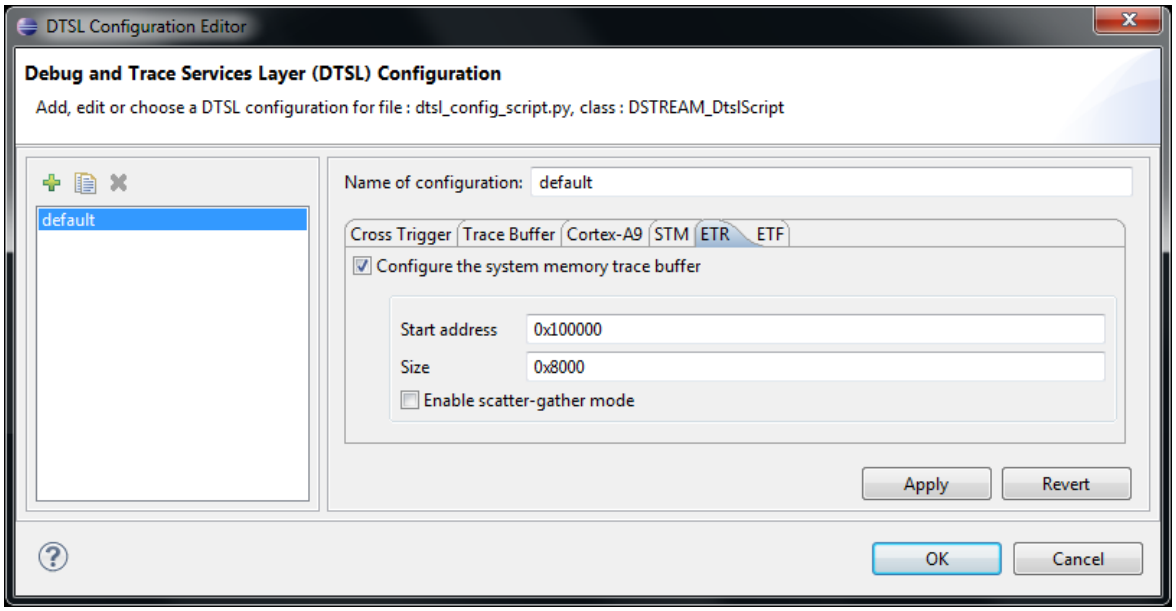
- **Enable STM Trace** – check to enable STM tracing.

ETR Settings

The **ETR** settings allow the configuration of the ETR (Embedded Trace Router) settings.

The Embedded Trace Router is used to direct the tracing information to a memory buffer accessible by HPS.

Figure 6-20: DTSL Configuration Editor - ETR



The following options are available:

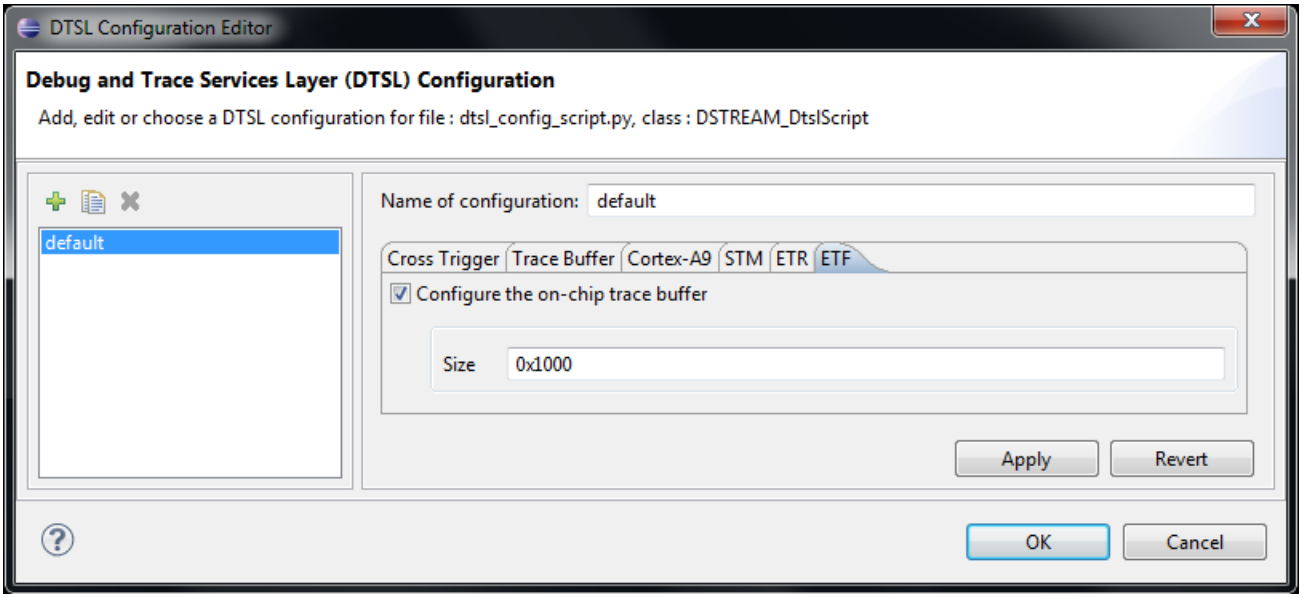
- **Configure the system memory trace buffer** – check this if the **ETR** was selected as the trace destination on the **Trace Buffer** tab
- **Start Address, Size** – define the trace buffer location in system memory, and its size
- **Enable scatter-gather mode** – used when the OS cannot guarantee a contiguous piece of physical memory. The scatter-gather table is setup by the operating system using a device driver, and is read automatically by the ETR.

ETF Settings

The **ETF** tab allows the configuration of the ETF (Embedded Trace FIFO) settings.

The Embedded Trace FIFO is a 32KB buffer residing on HPS that can be used to store tracing data to be retrieved by the debugger, but also as an elastic buffer for the scenarios where the tracing data is stored in memory through ETR or on the external DSTREAM device using TPIU.

Figure 6-21: DTSL Configuration Editor - ETF



The following options are available:

- **Configure the on-chip trace buffer** – check this if ETF was selected as the trace destination on the **Trace Buffer** tab.
- **Size** – define the ETF size. By default it is set up to 0x1000 (4KB) but it can be set to 0x8000 (32KB) to match the actual buffer size.

Embedded Command Shell 7

Subscribe Send Feedback

The Embedded Command Shell enables the user to invoke the SoC EDS tools without qualifying them with the full path. That is, commands like 'eclipse' or '`bsp-editor`' or '`arm-none-eabi-gcc`' can be executed directly.

On Windows, the Embedded Command Shell is started by running <*SoC EDS installation directory*>**\Embedded_Command_Shell.bat**.

On Linux, the Embedded Command Shell is started from the **Start** menu or by running <*SoC EDS installation directory*>**/embedded_command_shell.sh**.

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

HPS Preloader User Guide

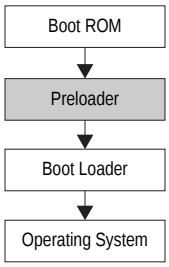
8

2013.05.03

Subscribe Send Feedback

There are four stages of the hard processor system (HPS) booting process; the preloader is the second stage.

Figure 8-1: Typical Boot Flow



The Preloader configures the HPS component based on the information from the handoff folder, initializes the SDRAM and then loads the next stage of the boot process into SDRAM and passes control to it.

The preloader can directly load your final application for Bare Metal applications and simple RTOSes.

Typically, a boot ROM loads the preloader from a flash device into the on-chip RAM and executes the preloader. The preloader can also be executed directly from the FPGAmemory.

Related Information

- **Arria V Device Handbook: Booting and Configuration**
  For more information about the four stages of the HPS booting process, refer to the *Booting and Configuration* appendix in volume 3 of the *Arria V Device Handbook*.
- **Cyclone V Device Handbook: Booting and Configuration**
  For more information about the four stages of the HPS booting process, refer to the *Booting and Configuration* appendix in volume 3 of the *Cyclone V Device Handbook*.

HPS Configuration

The preloader performs the following steps to configure the HPS and load the next image in the boot process:

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

1. Configures the HPS pins (I/O configuration shift register [IOCSR] and pin multiplexing)
2. Configures the HPS phase-locked loops (PLLs) and clocking
3. Configures the external SDRAM
4. Loads the next image in the boot process, typically stored in a flash device such as the NAND flash memory, Secure Digital/MultiMediaCard (SD/MMC) flash memory, or the quad serial peripheral interface (QSPI) flash memory
5. Jumps to the next loaded boot image

Preloader Support Package Generator

The preloader support package generator provides you with an easy, safe, and reliable way to customize the preloader.

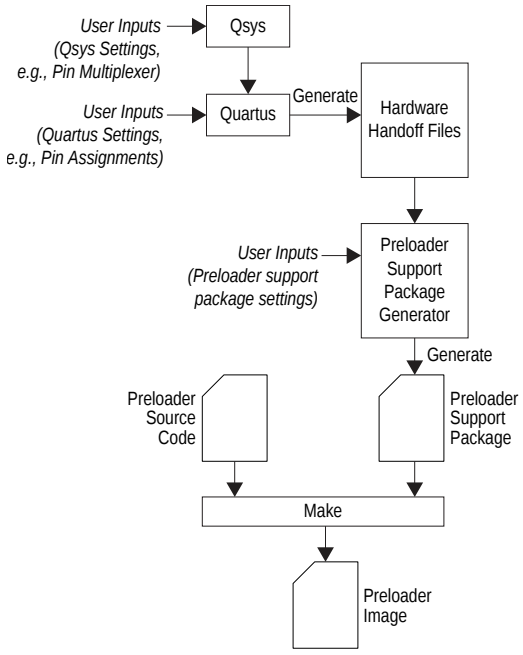
The preloader image tool creates an Altera boot ROM compatible image of the preloader.

The preloader support package generator creates a customized preloader support package with preloader generic source files and board-specific SoC FPGA files. The generator consolidates required hardware settings and your inputs to create the preloader support package. The support package files include a makefile to create the preloader image; you can download the preloader image to a flash device or FPGA RAM.

The preloader support package generator allows you to perform the following tasks:

- Create a new preloader support package
- Report preloader support package settings
- Modify preloader support package settings
- Generate preloader support package files

Figure 8-2: Preloader Support Package Generator Flow



Hardware Handoff Files

Use the Qsys system integration tool in the Quartus II software to generate a set of handoff files containing the hardware information required by the preloader.

The handoff files from the Qsys compilation are located in the *<quartus project directory>*/**hps_isw_handoff/<hps entity name>** directory (where **hps entity name** is the HPS component name in Qsys).

Note: You must update the hardware handoff files and regenerate the preloader support package each time a hardeare change impacts the HPS, such as after pin multiplexing or pin assignment.

Using the Preloader Support Package Generator GUI

You must perform the following steps to use the preloader support package generator GUI, **bsp-editor**:

1. Start an embedded command shell, as follows:
   - On a Windows-based system, run the batch file *<SoC EDS installation directory>***\Embedded_Command_Shell.bat**
   - On a Linux-based system, run the shell script *<SoC EDS installation directory>***/embedded_command_shell.sh**.
2. Run the **bsp-editor** command in the embedded command shell to launch the preloader support package generator.
3. To open and modify an existing preloader support package (PSP) project in the preloader support package generator, click **File** > **Open** and browse to an existing **.bsp** file.
4. To create a new PSP project, click **File** > **New BSP** to open the **New BSP** dialog box. The **New BSP** dialog box includes the following settings and parameters:
   - **Preloader settings directory** - the path to the hardware handoff files. The generator inspects the handoff files to verify the validity of this path
   - **Operating system** and **Version** - Not applicable to preloader generation.
   - **BSP target directory** - the destination folder for new BSP files created by the generator. This document refers to the preloader BSP directory as **<bsp_directory>**. The default directory name is **spl_bsp**. You can modify the directory name.
   - **BSP settings file name** - the location and filename of the **.bsp** file
   - **Additional .tcl scripting** - the location and filename of a **.tcl** script for overriding the default BSP settings
5. You can customize the PSP. After creating or opening a **.bsp** file, access the Settings in the **BSP Editor** dialogue box. The **Settings** are divided into **Common** and **Advanced** settings. When you select a group of settings, the controls for the selected settings appear on the right side of the dialogue box. When you select a single setting, the setting name, description and value are displayed. You can edit these settings in the **BSP Editor** dialogue box.
6. Click **Generate** to generate the preloader support package.
7. Click **Exit** to exit the preloader support package generator.

Using .tcl Scripts

Instead of using the default settings, you can create a tcl script file (**.tcl**) to define custom settings during BSP creation.

`set_setting` is the only available **.tcl** command. Refer to *BSP Settings* for a list of available settings.

Example 8-1: Valid .tcl Scripting Commands for Changing BSP Settings

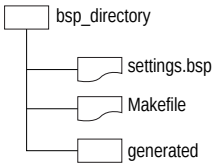
The following commands are used to set parameters n the BSP setings file:

```
set_setting spl.boot.BOOT_FROM_QSPI true
set_setting spl.boot.QSPI_NEXT_BOOT_IMAGE 0x50000
```

Preloader Support Package Files and Folders

The files and folders created with the preloader support package are stored in the location you specified in **BSP target directory** in the **New BSP** dialog box.

Figure 8-3: PSP Directory



The BSP files include:

- **settings.bsp** – the settings file containing all BSP settings
- **Makefile** – the makefile to create the preloader image; for more information, refer to *Preloader Compilation*
- **generated** – this folder contains files generated from the hardware handoff files from the Qsys system integration tool

Command-Line Tools for the Preloader Support Package Generator

The BSP command-line tools can be invoked from the embedded command shell, and provide all the features available in the preloader support package generator:

- The **bsp-create-settings** tool creates a new BSP settings file
- The **bsp-update-settings** tool updates an existing BSP settings file
- The **bsp-query-settings** tool reports the setting values in an existing BSP settings file
- The **bsp-generate-files** tool generates a BSP from the BSP settings file

Note: Help for each tool is available from the embedded command shell. To display help, type the following command:

```
<name of tool> --help
```

bsp-create-settings

The **bsp-create-settings** tool creates a new PSP settings file with default settings. You have the option to modify the BSP settings or generate the PSP files as shown in the following example.

Example 8-2: Creating a New PSP Settings File

```
bsp-create-settings --type spl --bsp-dir . \
   --settings settings.bsp \
   --preloader-settings-dir
../../hps_isw_handoff/<hps_entity_name>
```

Table 8-1: User Parameters: bsp-create-settings

Option	Required	Description
`--type <bsp-type>`	Yes	This option specifies the type of BSP. "spl" is the only allowed PSP type for a SoC EDS PSP.
`--settings <settings-file>`	Yes	This option specifies the path to a BSP settings file. The file is created with default settings. Altera recommends that you name the BSP settings file "settings.bsp".
`--preloader-settings-dir <preloader-settings-dir>`	No	This option specifies the path to the hardware handoff files.
`--bsp-dir <bsp-dir>`	No	This option specifies the path where the BSP files are generated. When specified, **bsp-create-settings** generates the files after the settings file has been created. Altera recommends that you always specify this parameter with **bsp-create-settings**.
`--set <name> <value>`	No	This option sets the BSP setting <name> to the value <value>. Refer to *BSP Settings* for a complete list of available setting names and descriptions.

Related Information

BSP Settings

For a complete list of available setting names and descriptions, refer to*BSP Settings*.

bsp-update-settings

The **bsp-update-settings** tool updates the settings stored in the BSP settings file, as shown in the following example.

Example 8-3: Updating a PSP

The following command changes the value of a parameter inside the file **"settings.bsp"**:

```
bsp-update-settings --settings settings.bsp --set \
   spl.debug.SEMIHOSTING 1
```

Table 8-2: User Parameters: bsp-update-settings

Option	Required	Description
`--settings <settings-file>`	Yes	This option specifies the path to an existing BSP settings file to update.
`--preloader-settings-dir <preloader-settings-dir>`	No	This option specifies the path to the handoff files.
`--bsp-dir <bsp-dir>`	No	This option specifies the path where the BSP files are generated. When this option is specified, **bsp-create-settings** generates the BSP files after the settings file has been created. Altera recommends that you specify this parameter with **bsp-create-settings**.
`--set <name> <value>`	No	This option sets the BSP setting <name> to the value <value>. Refer to *BSP Settings* for a complete list of available setting names and descriptions.

Related Information

BSP Settings

For a complete list of available setting names and descriptions, refer to*BSP Settings*.

bsp-query-settings

The **bsp-query-settings** tool queries the settings stored in BSP settings file, as shown in the following example. Setting values are sent to courier.

Example 8-4: Querying a PSP

The following command will retrieve all the settings from **"settings.bsp"** and displays the setting names and values.

```
bsp-query-settings --settings settings.bsp --get-all --show-names
```

Table 8-3: User Parameters: bsp-query-settings

Option	Required	Description
`--settings <settings-file>`	Yes	This option specifies the path to an existing BSP settings file.
`--get <name>`	No	This option instructs **bsp-query-settings** to return the value of the BSP setting <name>.
`--get-all`	No	This option shows all the BSP settings values. When using `--get-all`, you should also use `--show-names`.
`--show-names`	No	This option only takes effect when used together with `--get <name>` or `--get-all`. When used with one of these options, names and values of the BSP settings are shown side-by-side.

bsp-generate-files

The **bsp-generate-files** tool generates the files and settings stored in BSP settings file, as shown in the following examples.

Example 8-5: Generating Files After BSP Creation

The following command creates a settings file based on the handoff folder, then generates the Preloader source files based on those settings:

```
bsp-create-settings --type spl --bsp-dir . \
   --settings settings.bsp \
   --preloader-settings-dir \
   ../../hps_isw_handoff/<hps_entity_name>
bsp-generate-files --settings settings.bsp --bsp-dir
```

Example 8-6: Generating Files After BSP Updates

```
bsp-update-settings --settings settings.bsp --set \
   spl.debug.SEMIHOSTING 1
bsp-generate-files --settings settings.bsp --bsp-dir
```

Use the **bsp-generate-files** tool when BSP files need to be regenerated under one of the following conditions:

- **bsp-create-settings** creates the PSP, but the --bsp-dir parameter was not specified, so PSP files were not generated.
- **bsp-update-settings** updates the PSP, but the --bsp-dir parameter was not specified, so the files were not updated.
- You want to ensure the BSP files are up-to-date

Table 8-4: User Parameters: bsp-generate-files

Option	Required	Description
`--settings <settings-file>`	Yes	This option specifies the path to an existing BSP settings file.
`--bsp-dir <bsp-dir>`	Yes	This option specifies the path where the BSP files are generated.

BSP Settings

The preloader support package generator includes BSP settings for the following command options:

Table 8-5: Command Options

Command	Option
bsp-create-settings	`--set`
bsp-update-settings	`--set`
bsp-query-settings	`--get`
	`--get-all`
	`--show-names`

Table 8-6: Available BSP Settings

BSP Setting	Type	Default Value	Description
`spl.PRELOADER_TGZ`	String	**$(SOCEDS_DEST_ ROOT) /host_tools/ altera/ preloader/ uboot-socfpga.tar.gz**	This setting specifies the path to archive file containing the preloader source files.
`spl.CROSS_COMPILE`	String	**arm-none-eabi-**	This setting specifies the cross compilation tool chain for use.
`spl.boot.BOOT_FROM_QSPI`	Boolean	False	This setting loads the boot loader image from QSPI.
`spl.boot.BOOT_FROM_SDMMC`	Boolean	True	This setting loads the subsequent boot image from Secure Digital/ MultiMediaCard (SD/MMC).
`spl.boot.BOOT_FROM_RAM`	Boolean	False	This setting loads the subsequent boot image from RAM.
`spl.boot.QSPI_NEXT_BOOT_IMAGE`	Hexadecimal	0x60000	This setting specifies the location of the subsequent boot image in QSPI.

BSP Setting	Type	Default Value	Description
spl.boot.SDMMC_NEXT_BOOT_IMAGE	Hexadecimal	0x40000	This setting specifies the location of the subsequent boot image in SD/MMC.
spl.boot.WATCHDOG_ENABLE	Boolean	True	This setting enables the watchdog during the preloader execution phase. The watchdog remains enabled after the preloader exits.
spl.boot.CHECKSUM_NEXT_IMAGE	Boolean	True	This setting enables the preloader to validate the checksum in the subsequent boot image's header information.
spl.boot.EXE_ON_FPGA	Boolean	False	This setting executes the preloader on the FPGA. Select spl.boot.EXE_ON_FPGA when the preloader is configured to boot from the FPGA.
spl.boot.STATE_REG_ENABLE	Boolean	True	This setting enables writing the magic value to the INITSWSTATE register in the system manager when the preloader exists; this indicates to the boot ROM that the preloader has run successfully.
spl.boot.BOOTROM_HANDSHAKE_CFGIO	Boolean	True	This setting enables handshake with boot ROM when configuring the IOCSR and pin multiplexing. If spl.boot.BOOTROM_HANDSHAKE_CFGIO is enabled and warm reset occurs when the preloader is configuring IOCSR and pin multiplexing, the boot ROM will reconfigure IOCSR and pin multiplexing again. This option is enabled by default.

BSP Setting	Type	Default Value	Description
spl.boot.WARMRST_SKIP_CFGIO	Boolean	True	This setting enables the preloader to skip IOCSR and pin multiplexing configuration during warm reset. spl.boot.WARMRST_SKIP_CFGIO is only applicable if the boot ROM has skipped IOCSR and pin multiplexing configuration.
spl.debug.DEBUG_MEMORY_WRITE	Boolean	False	This setting enables the preloader to write debug information to memory for debugging, useful when UART is not available. The address is specified by spl.debug.DEBUG_MEMORY_ADDR
spl.debug.SEMIHOSTING	Boolean	False	This setting enables semihosting support in the preloader, for use with a debugger tool. spl.debug.SEMIHOSTING is useful when UART is unavailable. Refer to the *ARM Infocenter* for more information on semihosting.
spl.debug.HARDWARE_DIAGNOSTIC	Boolean	False	This setting enables hardware diagnostic support, enabling hardware to read from and write to the SDRAM to ensure hardware is working; the status is reported in the console.
spl.debug.SKIP_SDRAM	Boolean	False	The preloader skips SDRAM initialization and calibration when this setting is enabled.
spl.performance.SERIAL_SUPPORT	Boolean	True	This setting enables UART print out support, enabling preloader code to call printf() at runtime with debugging information. stdout output from printf() is directed to the UART. You can view this debugging information by connecting a terminal program to the UART specified peripheral.

BSP Setting	Type	Default Value	Description
spl.reset_assert.<peripheral_name>	Boolean	Refer to Table 8-7:spl.reset_assert table for peripheral names and values.	This setting forces the device to remain under reset state. You can include multiple instances of spl.reset_assert.<peripheral_name> to hold multiple peripherals in reset. You must ensure the debugger does not read registers from these components.
spl.warm_reset_handshake.FPGA	Boolean	True	This setting enables the reset manager to perform handshake with the FPGA before asserting a warm reset.
spl.warm_reset_handshake.ETR	Boolean	True	This setting enables the reset manager to request that the Embedded Trace Router (ETR) stalls the Advanced eXtensible Interface (AXI) master and waits for the ETR to finish any outstanding AXI transactions before asserting a warm reset of the L3 interconnect or a debug reset of the ETR.
spl.warm_reset_handshake.SDRAM	Boolean	True	This setting enables the reset manager to request that the SDRAM controller puts the SDRAM device into self-refresh mode before asserting warm reset.
spl.boot.FPGA_MAX_SIZE	Hexadecimal	0x10000	This setting specifies the maximum code (.text and .rodata) size that can fit within the FPGA. If the code build is bigger than the specified size, a build error is triggered. (1)
spl.boot.FPGA_DATA_BASE	Hexadecimal	0xffff0000	This setting specifies the base location for the data region (.data, .bss, heap and stack) when execute on FPGA is enabled.

(1) .text and .rodata are default memory sections defined by the linker tool in the GCC tool chain.
(2) .data and .bss are default memory sections defined by the linker tool in the GCC tool chain.

BSP Setting	Type	Default Value	Description
spl.boot.FPGA_DATA_MAX_SIZE	Hexadecimal	0x10000	This setting specifies the maximum data (.data, .bss, heap and stack) size that can fit within FPGA. If the code build is bigger than the specified size, a build error is triggered.
spl.debug.DEBUG_MEMORY_ADDR	Hexadecimal	0xfffffd00	This setting specifies the base address for storing preloader debug information enabled with the spl.debug.DEBUG_MEMORY_WRITE setting.
spl.debug.DEBUG_MEMORY_SIZE	Hexadecimal	0x200	This setting specifies the maximum size used for storing preloader debug information.

#unique_80/unique_80_Connect_42_table_950CF11383D54E11A719D728E572F211 lists the available variations of the spl.reset_assert.<*peripheral_name*> setting with their default values. Refer to #unique_80/unique_80_Connect_42_table_3B1A66707E3946F4B4FAAC146A63D9BB for a general description of the spl.reset_assert.<*peripheral_name*> setting.

Table 8-7: spl.reset_assert.<*peripheral_name*>

BSP Setting	Default Value
spl.reset_assert.DMA	False
spl.reset_assert.FPGA_DMA0	True
spl.reset_assert.FPGA_DMA1	True
spl.reset_assert.FPGA_DMA2	True
spl.reset_assert.FPGA_DMA3	True
spl.reset_assert.FPGA_DMA4	True
spl.reset_assert.FPGA_DMA5	True
spl.reset_assert.FPGA_DMA6	True
spl.reset_assert.FPGA_DMA7	True
spl.reset_assert.GPIO0	False
spl.reset_assert.GPIO1	False
spl.reset_assert.GPIO2	False
spl.reset_assert.L4WD1	False
spl.reset_assert.OSC1TIMER1	False
spl.reset_assert.SDR	False

BSP Setting	Default Value
spl.reset_assert.SPTIMER0	False
spl.reset_assert.SPTIMER1	False

Preloader Compilation

The makefile created by the PSP generator compiles the preloader sources and generates a preloader image. The makefile performs the following tasks:

- Copies the generic preloader source code into <*bsp_directory*>**/uboot-socfpga**
- Copies the generated BSP files and hardware handoff files to the source directory in <*bsp_directory*>**/uboot-socfpga/board/altera/socfpga_<*device*>**
- Configures the compiler tools to target an SoC FPGA
- Compiles the source files in <*bsp directory*>**/uboot-socfpga** with the user-specified cross compiler (specified in the BSP settings) and stores the generated preloader binary files in <*bsp directory*>**/uboot-socfpga/spl**
- Converts the preloader binary file to a preloader image, <*bsp directory*>**/preloader-mkpimage.bin**, with the **mkpimage** tool

The **mkpimage** tool is part of the preloader image tool. It inserts the correct header information and creates an Altera boot-ROM compatible image of the preloader. You can run the **make** utility in the command shell to compile the preloader in the BSP directory. The makefile contains the following targets:

- make all – compiles the preloader
- make clean – deletes **preloader-mkpimage.bin** from ***<bsp_directory>***
- make clean-all – deletes <*bsp directory*>, including the source files in the directory

Configuring FPGA with Image from QSPI Flash

The Preloader has the ability to configure the FPGA by using a configuration file stored in QSPI Flash.

In order to use this feature, the following steps need to be performed:

1. Configure the Preloader load the next boot stage from QSPI. Next boot stage image has to reside in QSPI Flash in this case.
2. Generate Preloader.
3. Compile Preloader, to make sure all the source code is available.
4. Modify file **uboot-socfpga/include/configs/socfpga_common.h** to have the macro CONFIG_SPL_FPGA_LOAD defined. It is undefined by default.
5. Recompile Preloader.
6. Convert SOF file to RBF using Quartus Programmer.
7. Convert RBF file to an image file using the mkimage utility.
8. Program the image file to QSPI at the address CONFIG_SPL_FPGA_QSPI_ADDR, which is defined in **uboot-socfpga/include/configs/socfpga_common.h**.
9. Boot board.

Loading Next Image from FAT Partition on SD Card

The Preloader has the ability to load the next boot stage image from a FAT partition on an SD card.

This is an alternative to putting the next boot image on the custom SD card partition with ID=0xA2. It is more convenient to write data to a FAT partition than a custom partition, especially on a Windows host OS.

In order to use this feature, the following steps need to be performed:

1. Configure the Preloader load the next boot stage from SD/MMC.
2. Generate Preloader.
3. Compile Preloader, to make sure all the source code is available.
4. Modify file **uboot-socfpga/include/configs/socfpga_common.h** to have the macro CONFIG_SPL_FAT_SUPPORT defined. It is undefined by default.
5. In the same file, edit the following macros if necessary:
   - #define CONFIG_SYS_MMC_SD_FAT_BOOT_PARTITION 1
   - #define CONFIG_SPL_FAT_LOAD_PAYLOAD_NAME "u-boot.img"
6. Copy the next stage boot image on the SD card FAT partition.
7. Boot board.

Preloader Image Tool

The preloader image tool creates an Altera boot-ROM compatible image of the preloader. The tool can also decode the header of previously generated images.

The preloader image tool makes the following assumptions:

1. The input file format is raw binary. You must use the `objcopy` utility provided with the GNU Compiler Collection (GCC) tool chain from the Mentor Graphics website (www.mentor.com) to convert other file formats, such as Executable and Linking Format File (**.elf**), Hexadecimal (Intel-Format) File (**.hex**), or S-Record File (**.srec**), to a binary format. The output file format is binary.
2. The preloader image tool always creates the output image at the beginning of the binary file. If the image must be programmed at a specific base address, you must supply the address information to the flash programming tool.
3. The output file contains only preloader images. Other images such as Linux, SRAM Object File (**.sof**) and user data are programmed separately using a flash programming tool or related utilities in the U-boot on the target system.

Related Information

Mentor Graphics

For more information about the GNU Compiler Collection (GCC) toolchain, refer to the Mentor Graphics website.

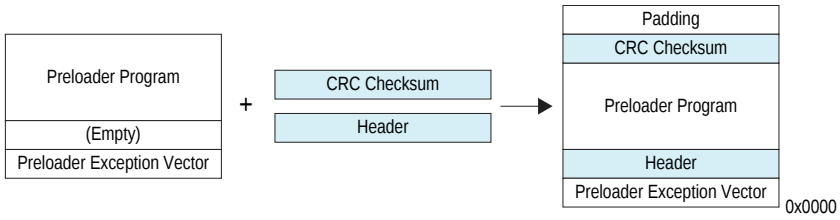
Operation of the Preloader Image Tool

The preloader image tool runs on a host machine. The tool generates the header and CRC checksum and inserts them into the final preloader image with the preloader program image and preloader exception vector.

For certain flash memory tools, the position of the preloader images must be aligned to a specific block size; the preloader image tool generates any padding data that may be required.

The preloader image tool optionally decodes and validates header information when given a pre-generated preloader image.

Figure 8-4: Basic Operation of the Preloader Image Tool

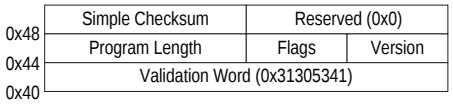


As illustrated, the binary preloader image is an input to the preloader image tool. The compiler leaves an empty space between the preloader exception vector and the program. The preloader image tool overwrites this empty region with header information and calculates a checksum for the whole image. When necessary, the preloader image tool appends the padding data to the output image.

The header includes:

- Validation word
- Version field (set to 0x0)
- Flags field (set to 0x0)
- Program length measured by the number of 32 bit words in the preloader program
- 16 bit checksum of the header contents (0x40 – 0x49)

Figure 8-5: Header Format



Tool Usage

The preloader image tool has three usage models:

1. Single image creation
2. Quad image creation
3. Single or quad image decoding

If an error is found during the make image process, the tool stops and reports the error. Possible error conditions include:

- The input image size is equal to or less than 80 bytes
- The input image size is equal to or greater than 60 kilobytes (KB)

mkpimage invokes the preloader image tool; invoking the tool with the `--help` option provides a tool description and tool usage and option information.

```
> mkpimage --help
mkpimage version 13.1 (build 154)
© 2013  Altera Corporation
Description: This tool creates an Altera BootROM-compatible image of
Second
```

```
Stage Boot Loader (SSBL). The input and output files are in binary
format.
It can also decode and check the validity of previously generated image.

Usage:
Create quad image:
mkpimage [option(s)] -o <outfile> <infile> <infile> <infile> <infile>
Create single image:
mkpimage [option(s)] -o <outfile> <infile>
Decode single/quad image:
mkpimage -d [-a <num>] <infile>

Options:
-a (--alignment) <num> : Address alignment in kilobytes, valid value
starts
from 64, 128, 256 etc, default to 64, override if the NAND flash has
a larger block size
-d (--decode)          : Flag to decode the header information from
input
file and display it
-f (--force)          : Flag to force decoding even if the input file
 is
an

unpadded image
-h (--help)            : Display this help message and exit
-o (--output) <outfile> : Output file, relative and absolute path
supported
-v (--version) <num>   : Header version to be created, default to 0
```

Output Image Layout

Base Address

You should place the preloader image at 0x0 for NAND and QSPI flash. The SD/MMC flash has a MBR that points to a specific offset at the start of the partition. The partition is of type 0xA2, a custom raw partition type without any file system. The preloader image tool always places the output image at the start of the output binary file, regardless of the target flash memory type. The flash programming tool is responsible for placing the image at the desired location on the flash memory device.

Related Information

Booting and Configuration

For more information, refer to the *Booting and Configuration* appendix in volume 3 of the *Arria V Device Handbook* or the *Cyclone V Device Handbook*.

Size

A single preloader has a 60 KB image size. You can store up to four preloader images in flash. If the boot ROM does not find a valid preloader image at the first location, it attempts to read an image from the next location, 64 KB above the first. To take advantage of this feature, program four preloader images in flash at consecutive 64 KB intervals.

Related Information

Booting and Configuration
For information about how the boot ROM loads preloader images, see "Boot ROM Flow" in the*Booting and Configuration* appendix in volume 3 of the *Arria V Device Handbook* or the *Cyclone V Device Handbook*.

Address Alignment

Every preloader image aligns to a 64 KB boundary at offsets 0x0, 0x10000, 0x20000, and 0x30000, except for the NAND flash. Version 0 of the boot ROM assumes that all preloader images in flash memory align to 64 KB boundaries, except in the case of NAND flash.

If the preloader images are stored in NAND flash with an erasable block size larger than 64 KB, preloader images are aligned to the block size.

The preloader image tool is unaware of the target flash memory type. If you do not specify the block size, the default is 64 KB.

NAND Flash

Each preloader image occupies an integer number of blocks. A block is the smallest entity that can be erased, so updates to a particular boot image does not impact the other images. The size of a single preloader image sizing is either 64 KB or the NAND flash block size, whichever is larger. For example, if a NAND block is 32 KB or 64 KB, a single preloader image size is 64 KB; if a NAND block is 128 KB, a single preloader image size is 128 KB.

Serial NOR Flash

Each QSPI boot image occupies an integer number of sectors unless subsector erase is supported; this ensures that updating one image does not affect other images.

SD/MMC

The master boot record, located at the first 512 bytes of the device memory, contains partition address and size information. The preloader and U-boot images are stored in partitions of type 0xA2. Other images are stored in partition types according to the target file system format.

You can use the **fdisk** tool to set up and manage the master boot record. When the **fdisk** tool partitions an SD/MMC device, the tool creates the master boot record at the first sector, with partition address and size information for each partition on the SD/MMC.

Padding

The preloader image tool inserts a CRC checksum in the unused region of the image. Padding fills the remaining unused regions. The contents of the padded and unused regions of the image are undefined.

Related Information

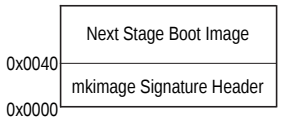
Booting and Configuration
For more information, refer to the *Booting and Configuration* appendix in volume 3 of the *Arria V Device Handbook* or the *Cyclone V Device Handbook*.

mkimage Tool

The preloader verifies the `mkimage` header appended to the boot image before the preloader loads the next stage boot image in the HPS booting process. The next stage boot image is a U-boot image, an RTOS, or a bare-metal application.

The preloader source files include the U-boot utility **mkimage** tool. The **mkimage** tool appends the `mkimage` header to the next image.

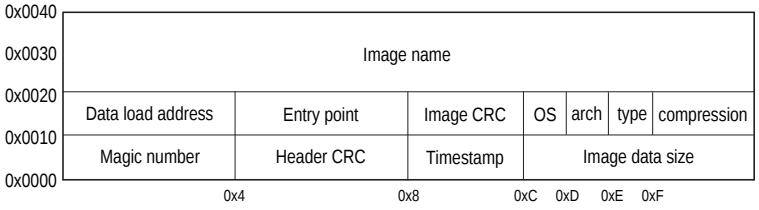
Figure 8-6: `mkimage` Header from mkimage Tools



The preloader reads the following information from `mkimage` header:

1. Image magic number - determines if the image is a valid boot image
2. Image data size - the length of the boot image to be copied
3. Data load address - the entry point of the boot image
4. Operating system - determines if the image is a U-boot image or another type of image
5. Image name - the name of the boot image
6. Image CRC - the checksum value of the boot image

Figure 8-7: `mkimage` Header Layout from mkimage Tools



The **mkimage** tool source files are included with the preloader source files in <*bsp_directory*>**/uboot-socfpga/tools/mkimage** . The **mkimage** tool compiles and generates during preloader compilation, but you can alternatively create the **mkimage** tool by typing `make tools` in ***<bsp_directory>***. To use the **mkimage** tool, you can add **<bsp_directory>/uboot-socfpga/tools** to the system PATH environment variable.

`mkimage` invokes the **mkimage** tool and the `--help` option provides the tool description and option information

mkimage Tool Options

The `--help` option of the **mkimage** tool provides the tool description and option information.

```
$ mkimage
Usage: mkimage -l image
          -l ==> list image header information
      mkimage [-x] -A arch -O os -T type -C comp -a addr -e ep -n name
 -d data_file[:data_file...] image
          -A ==> set architecture to 'arch'
          -O ==> set operating system to 'os'
          -T ==> set image type to 'type'
          -C ==> set compression type 'comp'
          -a ==> set load address to 'addr' (hex)
          -e ==> set entry point to 'ep' (hex)
```

```
   -n ==> set image name to 'name'
   -d ==> use image data from 'datafile'
   -x ==> set XIP (execute in place)
mkimage [-D dtc_options] -f fit-image.its fit-image
mkimage -V ==> print version information and exit
```

mkimage Tool Image Creation

Example 8-7: Creating a U-boot Image

```
mkimage –A arm –T firmware –C none –O u-boot –a 0x08000040 –e 0
–n “U-Boot 2011.12 for SOCFGPA board” –d u-boot.bin u-boot.img
```

Example 8-8: Creating a Bare-metal Application Image

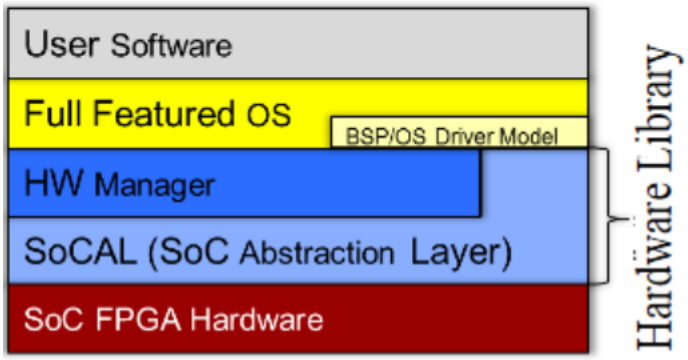
```
mkimage -A arm -O u-boot -T standalone -C none -a 0x02100000 -e
0 -n "baremetal image" -d hello_world.bin hello_world.img
```

Hardware Library Overview 9

Subscribe Send Feedback

The Altera SoC FPGA Hardware Library (HWLIB) was created to address the needs of low-level software programmers who require full access to the configuration and control facilities of SoC FPGA hardware. An additional purpose of the HWLIB is to mitigate the complexities of managing the operation of a sophisticated, multi-core application processor and its integration with hardened IP peripheral blocks and programmable logic in a SoC architecture.

Figure 9-1: HW Library



Within the context of the SoC HW/SW ecosystem, the HWLIB is capable of supporting software development in conjunction with full featured operating systems or standalone bare-metal programming environments. The relationship of the HWLIB within a complete SoC HW/SW environment is illustrated in the above figure.

The HWLIB provides a symbolic register abstraction layer known as the SoCAL that enables direct access and control of HPS device registers within the address space. This layer is a necessary enabler for several key stakeholders (boot loader developers, driver developers, board support package developers, debug agent developers, board bring-up engineers) requiring a precise degree of access and control of the hardware resources.

The HWLIB also deploys a set of Hardware Manager (HW Manager) APIs that provides more complex functionality and drivers for higher level use case scenarios.

The HWLIB has been developed as a source code distribution. The intent of this model is to provide a useful set of out-of-the-box functionality and to serve as a source code reference implementation that a user can tailor accordingly to meet their target system requirements.

The capabilities of the HWLIB are expected to evolve and expand over time particularly as common use case patterns become apparent from practical application in actual systems.

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

In general, the HWLIB assumes to be part of the system software that is executing on the Hard Processor System (HPS) in privileged supervisor mode and in the secure state.

The anticipated HWLIB clients include:

- Bare-Metal application developers
- Custom preloader and boot loader software developers
- Board support package developers
- Diagnostic tool developers
- Software driver developers
- Debug agent developers
- Board bring-up engineers
- Other developers requiring full access to SoC FPGA hardware capabilities

Feature Description

This section provides a description of the operational features and functional capabilities present in the HWLIB. An overview and brief description of the HWLIB architecture is also presented.

The HWLIB is a software library architecturally comprised of two major functional components:

- SoC Abstraction Layer (SoCAL)
- Hardware Manager (HW Manager)

SoC Abstraction Layer (SoCAL)

The SoC Abstraction Layer (SoCAL) presents the software API closest to the actual HPS hardware. Its purpose is to provide a logical interface abstraction and decoupling layer to the physical devices and registers that comprise the hardware interface of the HPS.

The SoCAL provides the benefits of:

- A logical interface abstraction to the HPS physical devices and registers including the bit-fields comprising them.
- A loosely coupled software interface to the underlying hardware that promotes software isolation from hardware changes in the system address map and device register bit field layouts.

Hardware Manager (HW Manager)

The Hardware Manager (HW Manager) component provides a group of functional APIs that address more complex configuration and operational control aspects of selected HPS resources.

The HW Manager functions have the following characteristics:

- Functions employ a combination of low level device operations provided by the SoCAL executed in a specific sequence to effect a desired operation.
- Functions may employ cross functional (such as from different IP blocks) device operations to implement a desired effect.
- Functions may have to satisfy specific timing constraints for the application of operations and validation of expected device responses.
- Functions provide a level of user protection and error diagnostics through parameter constraint and validation checks.

The HW Manager functions are implemented using elemental operations provided by the SoCAL API to implement more complex functional capabilities and services. The HW Manager functions may also be

implemented by the compound application of other functions in the HW Manager API to build more complex operations (for example, software controlled configuration of the FPGA).

Hardware Library Reference Documentation

Reference documentation for the SoCAL API and HW Manager API is distributed as part of the SoCEDS Toolkit. This reference documentation is provided as online HTML accessible from any web browser.

The locations of the online SoC FPGA Hardware Library (HWLIB) Reference Documentation are:

- SoC Abstraction Layer (SoCAL) API Reference Documentation:<*SoC EDS installation directory*>**/ip/altera/hps/altera_hps/doc/socal/html/index.html**
- Hardware Manager (HW Manager) API Reference Documentation: <*SoC EDS installation directory*>**/ip/altera/hps/altera_hps/doc/hwmgr/html/index.html**.

HPS Flash Programmer User Guide 10

2013.05.03

Subscribe Send Feedback

The Altera Quartus II software and Quartus II Programmer include the hardware processor system (HPS) flash programmer. Hardware designs such as HPS incorporate flash memory on the board to store FPGA configuration data or HPS program data. The HPS flash programmer programs the data into a flash memory device connected to an Altera SoC. The programmer sends file contents over an Altera download cable, such as the USB-Blaster™ II, to the HPS and instructs the HPS to write the data to the flash memory.

The HPS flash programmer programs the following content types to flash memory:

- HPS software executable files — Many systems use flash memory to store non-volatile program code, or firmware. HPS systems can boot from flash memory.

  Note: The HPS Flash Programmer is mainly intended to be used for programming the Preloader Image to QSPI or NAND Flash. Due to the low speed of operation it is not recommended to be used for programming large files.

- FPGA configuration data — At system power-up, the FPGA configuration controller on the board or HPS reads FPGA configuration data from the flash memory to program the FPGA. The configuration controller or HPS may be able to choose between multiple FPGA configuration files stored in flash memory.
- Other arbitrary data files — The HPS flash programmer programs a binary file to any location in a flash memory for any purpose. For example, a HPS program can use this data as a coefficient table or a sine lookup table.

The HPS flash programmer programs the following memory types:

- Quad serial peripheral interface (QSPI) Flash
- ONFI (Open NAND Flash Interface) compliant NAND Flash

HPS Flash Programmer Command-Line Utilities

You can run the HPS flash programmer directly from the command line. For the Quartus II software, the HPS flash programmer is located in <*Altera installation directory*>**/quartus/bin**. For the Quartus II Programmer, the HPS flash programmer is located in <*Altera installation directory*>**/qprogrammer/bin**.

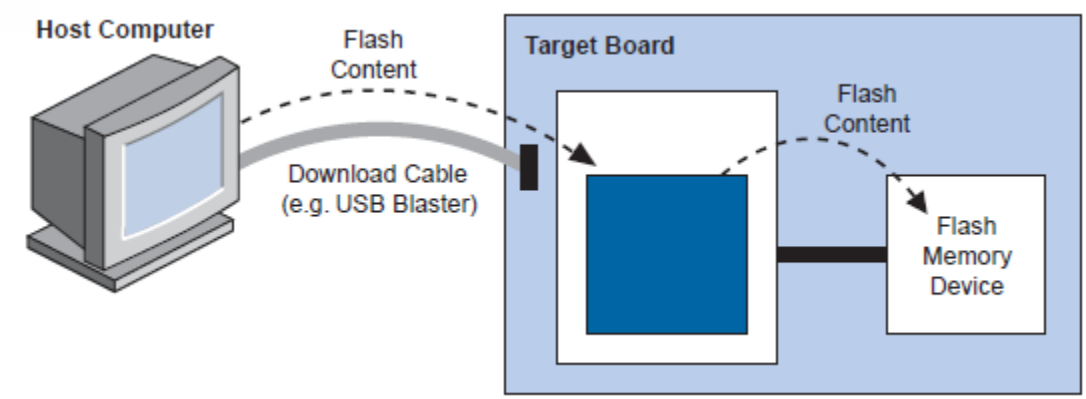
© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

How the HPS Flash Programmer Works

The HPS flash programmer is divided into a host and a target. The host portion runs on your computer and sends flash programming files and programming instructions over a download cable to the target. The target portion is the HPS in the SoC. The target accepts the programming data flash content and required information about the target flash memory device sent by the host. The target writes the data to the flash memory device.

Figure 10-1: HPS Flash Programmer



The HPS flash programmer determines the type of flash to program by sampling the boot select (BSEL) pins during cold reset; you do not need to specify the type of flash to program.

Using the Flash Programmer from the Command Line

HPS Flash Programmer

The HPS flash programmer utility can erase, blank-check, program, verify, and examine the flash. The utility accepts a Binary File with a required ".bin" extension.

The HPS flash programmer command-line syntax is:

```
quartus_hps <options> <file.bin>
```

Note: The HPS flash programmer uses byte addressing.

Table 10-1: HPS Flash Programmer Parameters

Option	Short Option	Required	Description
`--cable`	`-c`	Yes	This option specifies what download cable to use. To obtain the list of programming cables, run the command "jtagconfig". It will list the available cables, like in the following example: jtagconfig 1) USB-Blaster [USB-0] 2) USB-Blaster [USB-1] 3) USB-Blaster [USB-2] The "-c" parameter can be the number of the programming cable, or its name. The following are valid examples for the above case: -c 1 -c "USB-Blaster [USB-2]"
`--device`	`-d`	Yes (if there are multiple HPS devices in the chain)	This option specifies the index of the HPS device. The tool will automatically detect the chain and determine the position of the HPS device; however, if there are multiple HPS devices in the chain, the targeted device index must be specified.
`--operation`	`-o`	Yes	This option specifies the operation to be performed. The following operations are supported: • I: Read IDCODE of SOC device and discover Access Port • S: Read Silicon ID of the flash • E: Erase flash • B: Blank-check flash • P: Program flash • V: Verify flash • EB: Erase and blank-check flash • BP: Program <*BlankCheck*> flash • PV: Program and verify flash • BPV: Program (blank-check) and verify flash • X: Examine flash **Note:** The program begins with erasing the flash operation before programming the flash by default.
`--addr`	`-a`	Yes (if the start address is not 0)	This option specifies the start address of the operation to be performed.

Option	Short Option	Required	Description
`--size`	`-s`	No	This option specifies the number of bytes of data to be performed by the operation. `size` is optional.
`--repeat`	`-t`	No	These options should be used together. The HPS BOOT flow supports up to four images where each image is identical and these options duplicate the operation data; therefore you do not need eSW to create a large file containing duplicate images. `repeat` specifies the number of duplicate images for the operation to perform. `interval` specifies the repeated address. The default value is 64 kilobytes (KB). `repeat`and `interval`are optional.
`--interval`	`-i`		

HPS Flash Programmer Command Line Examples

Please run "quartus_hps --help" to obtain information about usage. You can also run "quartus_hps --help=<option>" to obtain more details about each option. For example "quartus_hps --help=o".

Example 10-1: quartus_hps –c 1 –o P input.bin

This example programs the input file (**input.bin**) into the flash, starting at flash address 0 using a cable *M*.

Example 10-2: quartus_hps –c 1 –o PV –a 1024 –s 500 input.bin

This example programs the first 500 bytes of the input file (**input.bin**) into the flash, starting at flash address 1024, followed by a verification using a cable *M*.

Example 10-3: quartus_hps –c 1 –o PV –a 0x400 –s 500 input.bin

This example programs the first 500 bytes of the input file (**input.bin**) into the flash, starting at flash address 1024, followed by a verification using a cable *M*. Without the prefix 0x for the flash address, the tool assumes it is decimal. With the prefix 0x, the tool assumes it is hexadecimal.

Example 10-4: quartus_hps –c 1 –o BPV –t 2 –i 0x100000 input.bin

This example programs the input file (**input.bin**) into the flash, using a cable *M*. The operation repeats itself twice at every 1 megabyte (MB) of the flash address. Before the program operation, the tool ensures the flash is blank. After the program operation, the tool verifies the data programmed.

Example 10-5: quartus_hps –c 1 –o EB input.bin

This example erases the flash on the flash addresses where the input file (**input.bin**) resides, followed by a blank-check using a cable *M*.

Example 10-6: quartus_hps –c 1 –o E

This example erases the full chip, using a cable *M*. When no input file (**input.bin**) is specified, it will erase all the flash contents.

Example 10-7: quartus_hps –c 1 –o E –a 0x100000 –s 0x400000

This example erases specified memory contents of the flash. For example, 4 MB worth of memory content residing in the flash address, starting at 1 MB, are erased using a cable *M*.

Example 10-8: quartus_hps –c 1 –o X –a 0x98679 –s 56789 output.bin

This example examines 56789 bytes of data from the flash with a 0x98679 flash start address, using a cable *M*.

Supported Memory Devices

Table 10-2: QSPI Flash

Manufacturer	Device ID	DIE #	Density (Mb)
Micron	0x18BA20	1	128
Micron	0x19BA20	1	256
Micron	0x20BA20	2	512
Micron	0x21BA20	4	1024
Micron	0x18BB20	1	128
Micron	0x19BB20	1	256
Micron	0x20BB20	2	512
Spansion	0x182001	1	128 (Sector size of 64 KB)
Spansion	0x182001	1	128 (Sector size of 256 KB)
Spansion	0x190201	1	256 (Sector size of 64 KB)

Manufacturer	Device ID	DIE #	Density (Mb)
Spansion	0x190201	1	256 (Sector size of 256 KB)
Spansion	0x200201	1	512

Table 10-3: ONFI Compliant NAND Flash

Manufacturer	MFC ID	Device ID	Density (Gb)
Micron	0x2C	0x48	16
Micron	0x2C	0xA1	8
Micron	0x2C	0xF1	8

Linux Compiler 11

Subscribe Send Feedback

The Linux compiler that is shipped with SoCEDS is the Linaro Linux compiler version 4.7.3.

The compiler is a GCC based **arm- linux - gnueabihf -** port. That is, it target the ARM processor, it assumes the target platform is running Linux, and it uses the GNU EABI HF conventions.

The Linux compiler is installed as part of the ARM DS-5 Altera Edition, which is installed as part of SoCEDS. The compilation tools are located in the following folder: <*SoC EDS installation directory*> **/ds-5/bin**.

The Linux compiler comes with full documentation, located in the following folder: <*SoC EDS installation directory*>**/ds-5/documents/gcc**. The documents are provided as HTML files. Some of the provided documents are:

- Compiler manual
- Assembler manual
- Linker manual
- Binutils manual
- GDB manual
- Getting Started Guide

Related Information

Linaro

For more information about the Linux compiler and for downloading the latest version of the tools, if necessary, refer to the download page at the Linaro website (www.linaro.org).

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Yocto Plugin 12

Subscribe Send Feedback

The Yocto Linux Source Package available on the Yocto Project website (www.yoctoproject.org) allows the entire Linux software stack (Kernel, drivers, device tree, root file system) targeting the SoC to be built in a very simple and convenient way.

The Yocto Eclipse plugin fulfills the need of the application developers to be able to target the Linux software stack without requiring them to learn the details on how to build the system. This enables the developers to focus on what they know best - developing applications.

The Yocto Eclipse plugin is installed on top of the ARM DS-5 Altera Edition. Documentation on how to install and use the Yocto Plugin is located on the Rocketboards website (www.rocketboards.org).

Related Information

- **Yocto Project**
  For more information about the Yocto Linux source project, refer to the Yocto Project website (http://www.yoctoproject.org).
- **Yocto Eclipse Plugin**
  For more information about the Yocto Eclipse Plugin, refer to the Rocketboards website (http://www.rocketboards.org).

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Device Tree Generator 13

Subscribe Send Feedback

A Device Tree is a data structure that describes the underlying hardware to an operating system - primarily Linux. By passing this data structure to the OS kernel, a single OS binary may be able to support many variations of hardware. This flexibility is particularly important when the hardware includes an FPGA.

The Device Tree Generator tool is part of Altera SoC EDS and is used to create device trees for SoC systems that contain FPGA designs created using Qsys. The generated Device Tree describes the HPS peripherals, selected FPGA Soft IP and also peripherals that are board dependent.

Related Information

- **Altera Wiki**
  For more information about DTG, refer to the Altera Wiki website.
- **Device Tree Generator User Guide**
  For more details, refer to the Device Tree Generator User Guide located on the Device Tree Generator Documentation page on the Rocketboards website.

© 2013 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered